Contents

Reports & Opinions

Dialogue

Features

Reviews

Copy Dates

C Vu 18.2: March 1st 2006

C Vu 18.3: May 1st 2006

Contact Information:

Editorial: Paul Johnson
77 Station Road, Haydock,
St Helens,
Merseyside, WA11 0JL
cvu@accu.org

Advertising: Thaddaeus Frogley
ads@accu.org

Treasurer: Stewart Brodie
29 Campkin Road,
Cambridge, CB4 2NL
treasurer@accu.org

ACCU Chair: Ewan Milne
0117 942 7746
chair@accu.org

Secretary: Alan Bellingham
01763 248259
secretary@accu.org

Membership Secretary: David Hodge
01424 219 807
membership@accu.org

Cover Art: **Alan Lenton**
Repro: **Parchment (Oxford) Ltd**
Print: **Parchment (Oxford) Ltd**
Distribution: **Able Types (Oxford) Ltd**

Membership fees and how to join:

Basic (C Vu only): £25

Full (C Vu and Overload): £35

Corporate: £120

Students: half normal rate

ISDF fee (optional) to support Standards work: £21

There are 6 issues of each journal produced every year.

Join on the web at www.accu.org with a debit/credit card, T/Polo shirts available.

Want to use cheque and post - email membership@accu.org for an application form.

Any questions - just email membership@accu.org

Reports & Opinions

Editorial

Paul F. Johnson <cvu@accu.org>

Before I start, may I wish everyone reading this edition a warm welcome to 2006. Okay, I know by the time you'll be reading this, it will be February with thoughts of the Conference just around the corner and where to get the best beer and curries from while there, but it's 2nd Jan in Haydock, so Happy New Year folks!

Bad Manners

We live in a world where most large organisations have access to such simple things like email, word processors, spreadsheets and computers in general. We have faxes, we have a plethora of instant messenger systems, we have VoIP and any number of other modes for communication; it has never been so easy to say "hi" to someone a thousand miles away at little or no real cost. Even the computer "geeks" of days gone by have shed their train spotter type image with this positive boon in communication as there is now no excuse for being a recluse, sitting in a room away from other humans and generally being unsociable.

In Autumn 2005, I embarked on searching for a new job (I'm no longer happy where I am and would even consider moving from my beloved Merseyside for the chance to do something which doesn't fill me with dread for facing another day). I have plenty of copies of my CV and am on quite a few recruiters mailing lists and cardexs around Manchester. I also look closely for any teaching positions having qualified last summer for anything post 16.

The applications began in earnest and I started to notice something on all application forms (and also when dealing with agencies) – they very rarely tell you anything about the application progress and take the approach that if you hear nothing, you get nowhere. It is even worse when you have that sort of treatment from your own employer!

There is nothing inherently wrong – it saves them time, but I doubt for one second that they even bothered to think the effect that has on applicants. It shows, in my opinion, nothing more than a complete disrespect for the applicant and makes the applicant's self esteem drop as not only have they not heard anything, but more over, any form of feedback.

Now, I'm not saying that every applicant should have a 5 minutes conversation with the HR departments or agency bod, but how much effort would it be to set up a list on whatever email client they use to send out a bulk email saying "thanks, but no thanks" and for those on the short-list, a more detailed email saying where they went wrong or how to improve next time around.

Of course, not everyone applying for jobs to organisations (rather than through agencies) has email and so the standard line of providing a SAE is fine. The point is, despite the voluminous number of ways to communicate, HR people and agencies seem not to be able to create an email list or talk directly to a candidate. That, to me, smacks of sheer bad manners and is probably one of the reasons why people shrug their shoulders after applying for goodness knows how many jobs and resign themselves to working in a place, completely demotivated, and because of that not being bothered over the quality of their performance.

Just for fun, I created a list last night from the ACCU manual of 200 random email addresses in Evolution. It didn't take long and if they had been added as applications arrived, it would have been a very trivial matter. This negates any sort of "too much work involved" excuse.

Agencies are even worse though. One moment they're all nice, the next, complete silence. Alright, I know that agencies are there to make money and do so by just passing on details, but that's still no real excuse for the lack of respect they seem to pay to prospective employees of the companies they represent. Don't get me wrong, I actually know quite a few people who work for agencies, and they are mostly really nice, but it does grate.

Does your CV Suck?

It seems that CV writing changes as often as most people change their socks! My original was about 10 pages long, which (when I was taught how to do it in the late 1980s) was the right way of doing it. Then it was down to a max. of 4 pages. Then 2 pages, up to 3 pages and then at a stretch 4 again (as long as it's really required).

Content also changed. Originally, personal details > qualifications (chronological order) > employment > extra > referees. Then the qualifications moved, employment was set out differently, emphasis on aspects moved, "extra" was dropped, brought back in, moved, changed, removed and finally placed back with a big red bow around it.

I think it's time CV writing was submitted to ECMA for documenting and standardising, together with application forms.

Don't Get Me Started on Application Forms!

Oh, okay then...

One thing which has interested me is the variation in information required on application forms from different employers. Not necessarily from different sectors, but from different organisations. Around St. Helens, there are a large number of Colleges of Further Education, and for the Merseyside ones, I've either been a pupil of or an employee of them all. About a year back, one a bit further out advertised for a position so I downloaded the application form and read through it. Now, the form was pretty standard for the most part. Where it differed was in the health section.

It was so intrusive that I refused to fill that section in other than for the pertinent aspects of my health (believe it or not, they wanted to know about my parents' health as well). I also enclosed with the application a note detailing why I had failed to fill it in and where they had gone over the mark with respect to confidentiality. Needless to say I heard nothing back, but it does highlight a problem – what is too far and what can a prospective employer ask?

In my case, because of a partial stomach collapse in 2000 and the NHS doing very little about it for five years, my attendance record at work was not that amazing (it is now a lot better – since my operation in March, I've had 2 days off and that was down to food poisoning) and so that has to go down on the application form. The employer doesn't bother looking at the reason, they just see that in 2004 - 2005, I had around 9 weeks off (6 was after the operation in recuperation).

Tell the Truth...

There is a post in Wakefield I'm considering applying for and I've hit a dilemma. The closing date is 2nd Feb which is about 8 months after my op. Do I turn around on the application and tell a little white lie (after all, the two days in 8 months is actually closer to what I would expect) and give 3 days as the total time off in the year or do I tell the truth and say 7 weeks, but just over 6 was due to my stomach complaint?

At the end of the day, I guess what I do is down to what I can live with combined with what I can reasonably get away with, so if I ignore that little voice on my shoulder, I think I can swing it.

Your Turn

Alright, time to tell the truth now. Other than those who work for themselves, who, in their entire employed life can stand up, put their hand on their heart and say that they have never embellished the truth on an application for a job – or better than that, never embellished and been appointed?

I'm pretty sure that everyone, at one time or another, has exaggerated the truth in order to get what they want. I'm not calling anyone a liar or anything like that, but it would be interesting to know. Well, I think so at least.

Don't Drink and Code

An interesting idea came to me over the holidays.

Every year, the Government runs a "don't drink and drive" campaign. I thought it would be a bit of fun to see if that would apply to programming and set about designing three tasks which should test me and also three different levels of intoxication.

Beverage	Mistakes (per 100 lines code)	Crashes	Compile	Quality of code
Sober	2	1	3	Nice and clean
2 pints	3	0	4	Clean, but stringy
¾ l Martini	8	4	6	Don't call us
Coffee	3	1	4	Nice and clean, but looks rushed in places

The tasks : debug some C++, write a small, text base calculator which takes everything from a text file on disc in C, design – on paper – a front end for my C#/MySQL series and then prototype it.

All three C languages are thereby covered.

The levels of being drunk : stone cold sober, 2 pints of Theakston's Old Peculiar and having drunk ¾ of a litre of Martini. For the tests, they were conducted 3 hours after eating.

The outcome is based on time to code, number of mistakes, number of crashes, code quality and number of times it takes to actually get the code to compile. It made for interesting reading – especially when compared to a 4th test based on consuming strong black coffee.

What is interesting is that in my, admittedly unscientific, test having a couple of pints yields code of roughly the same quality as when I drunk just coffee.

That's my excuse for not turning things in on time – you get your own!

Paul F Johnson

View From the Chair

Ewan Milne <chair@accu.org>

Happy New Year, and welcome to the first C Vu of 2006. Of course such seasonal greetings will be rather late by the time they reach you, but as I write it is the first week of January. It is freezing cold outside, and I am slotting together the final pieces of the conference programme, with the help as always of the conference committee. You will now (meaning as you read this) be able to see the programme for yourself at www.accu.org/conference, and will I am sure agree that, once more, we have a great line-up to offer. Our keynote speakers are Herb Sutter, Guido van Rossum, Helen Sharp and Hubert Matthews, and we are also featuring Michael Feathers, Peter Sommerlad, Jutta Eckstein and Nico Josuttis. Plus, hopefully, one or two very exciting names that I can't mention at the moment. As well as the return of the Python and Evolution of C++ tracks, we also focus on Distributed Collaboration, and Dynamic Languages.

On the Saturday of the conference we will as usual be holding the ACCU AGM. One of the main items on the agenda is the election of officers and committee. This will the first AGM under updated rules which allow nominations for posts to be lodged both in advance and from the floor at the meeting itself. See section 7 of the constitution for full details. If you wish to make a nomination in advance, please contact the Secretary at least 14 days before the meeting: that is by the 8th April.

As a reminder, the committee consists of the following officers:

- Chair
- Secretary
- Treasurer
- Membership Secretary
- Electronic Communications
- Publications
- Public Relations

In addition, there should be six or seven more committee members, depending on the number of members at the time of the meeting, plus several co-opted members. At the current time, all the officers have indicated that they are happy to continue in their posts. With one notable exception that is: as I indicated a little while ago (C Vu 17.2 in fact), I will be standing down as Chair. I am happy to report that at least one person has expressed a strong interest in taking over in this role. My intention is to remain a committee member, and looking ahead, to concentrate on preparations for the 2007 conference.

Ewan Milne

Membership Report

David Hodge <membership@accu.org>

At the end of 2005 we had 846 members, of which 109 were new members during 2005. We have members in 39 countries with the ones outside UK with over ten members being USA, Germany, Denmark, Australia, Switzerland, Sweden and The Netherlands. If you would like to reduce your membership fee by 5.00 pounds, then ask me for details on setting up a standing order.

I will be standing down from the Membership Secretary position at the 2007 AGM so if anyone would like to take over, or would like to find out what has to be done, please email me. It is intended to have a membership module on the website which will take care of most of the day to day operations.

Please address all queries on journals not received to me as I hold a small stock of spare journals.

David Hodge

Secretary's Report

Alan Bellingham <secretary@accu.org>

November, and now firmly back on track after the summer's languors, your committee met for the last time in 2005, in Nottingham at the house of Alan Griffiths.

As usual, we started with the reports, consideration of previous actions and the like. This included a quick overview of the forthcoming conference which is looking, yet again, very promising, with a full programme already.

After that, we had three items to consider.

The first is the redesign of these journals. We now have a new production editor (hi, Alison), and the page layout software being used is being

switched from, if I remember correctly, Quark to Adobe Framemaker. This would seem like an ideal point to refresh the layout: it may be more work than just changing tools, but if so, it's likely to be only a little more. However, the redesign needs some more input from the ACCU, and the switch will not happen until the redesign has been completed and signed off on. Hence, the issue you are reading will look like the previous ones.

Secondly, we had been asked to consider the Microsoft Safer C libraries. Now, this is a somewhat strange thing for your committee to talk about, as your representatives, but a motion had been raised and we had been asked to denounce the initiative as being detrimental to the C++ community. After some discussion, the consensus was reached that, misguided and regrettable as we as individuals might consider it, the committee does not have any moral right as a whole to have the society take a stance on this issue, but that individuals as individuals (and letter writers, article writers or even editors) should make their views known in these pages.

The third issue was the most important one for this society. As has previously been reported (both here and at the 2005 AGM), the ACCU web site is grievously out of date, and we have had plans to update it. These plans involved a company named Turtle Networks. Unfortunately, we have decided to part company with them on this project and pursue other options. Allan Kelly's report elsewhere covers this in more detail.

Our next meeting is currently scheduled for the 18th February, which will be our last before the AGM.

Alan Bellingham

Standards Report

Lois Goldthwaite <standards@accu.org>

My original intention for this column was to consider (1) whether C++ is already too complicated for its own or anyone else's good and (2) whether it is still worth spending time on C++ standardisation. To cut the debate short, the answers are (1) possibly, or possibly even probably (but there's more to it than that), and (2) yes (but there's more to it than that).

But current events suggest we should consider a different aspect of complication. At the time of writing, the C++ panel at BSI is studying a document which has been submitted for fast-track balloting to become an ISO/IEC standard. This standard from an ECMA Technical Group is called C++/CLI, and deals with extensions to standard C++ to enable writing powerful programs in a CLI environment. CLI stands for Common Language Infrastructure and is more commonly known as .Net from Microsoft.

At the time this effort was launched in 2003, participants described it as an attempt to develop a "binding" of C++ to CLI, and "a minimal (if still substantial) set of much-more-focused extensions" to support that environment, as compared with the earlier Managed C++ with its ugly __extended __keywords.

CLI, like Java, provides a virtual machine which interprets semi-compiled byte code and manages memory resources through garbage collection. Among its objectives are making it easier to combine modules written in different languages into a single application. It also aims to simplify the use of technologies such as XML, web services, and distributed programming.

Some of the fundamental decisions in the design of CLI show a different mindset from the traditional C/C++ programming model. Had I been the decider, some of them would probably have been different, but no doubt the authors had well-considered and defensible reasons for their choices. To achieve the objective of enabling C++ code to communicate with the CLI environment and its different model, certain – shall we call them – *adaptations* became necessary. One such is the invention of new keyword `gcnew` to allocate memory which will eventually be garbage collected. That doesn't look too unreasonable, as a platform-specific extension. Most compilers (g++ and Borland spring immediately to mind, also any tool for embedded programming) offer a certain number of extensions of their own, and people targeting the environments supported by those compilers often find them useful.

The group developing this "binding" called C++/CLI have been careful to guarantee that standard-conforming C++ code will compile and run correctly in this environment (though perhaps not taking full advantage of CLI's new features). This is much to be applauded.

HOWEVER, the BSI C++ panel opposes the plan to grant worldwide endorsement to C++/CLI as an international standard on the same par as ISO/IEC 14882 C++. The name is too similar, and the differences too immense. We believe C++/CLI has effectively evolved into a language which is almost, but not quite, entirely unlike C++ as we know it. And continuing to identify both languages by the same name (even with the all-too-often-dropped qualifier) will cause widespread confusion and damage to the industry and the standard language.

Now that is a strong statement, and is meant to be. C++/CLI adds at least two dozen new keywords to the 63 already in standard C++. It adds new syntax for some existing keywords (such as using `new` to indicate that a member function does not override an inherited one of the same name), and it gratuitously changes the meaning of various bits of currently-valid C++ syntax. The rules for determining which semantics should be invoked for any given line of code in a single translation unit are subtle and will add a very large amount of complexity to the intellectual burden of writing and understanding C++.

Perhaps a brief code example will illustrate why we are concerned.

```
// Example A - this is C++
class Base {
  virtual void f1( int i );
  int f2();
};

class Derived : Base {
  int x;
  public:
    void f1( string s );
    int f2(){ return x; };
};

// Example B - this is C++/CLI
interface class Base {
  virtual void f1( int i );
  int f2();
};

ref class Derived : Base {
  int x;
  public:
    void f1( string s );
    int f2( ) { return x; };
};
```

In Example A, the member functions of `Base` are (by default) private, concrete functions, and `f1()` is virtual but not `f2()`. In Example B, the only visible code difference is the new keyword interface, but (by default) all member functions are public, abstract, and virtual (even if not identified as such).

In Example A, because `Derived` has the keyword `class`, not `struct`, it has private inheritance from `Base` and therefore cannot be implicitly converted to `Base`. In example B, because `Base` is a CLI class and not a native C++ one, it confers public inheritance on `Derived`.

In C++, `Derived().f1( 42 );` would fail at compile time, because the declaration `Derived::f1( string );` hides the inherited function with the same name but different parameter. In C++/CLI, all inherited overloads are visible and callable; it is up to the programmer to beware whether she has unintentionally duplicated a name lurking much higher in the ancestry.

In C++/CLI, calling a virtual function in a constructor or destructor invokes the most-derived matching overload from a descendant, which may not be validly constructed when a base's `ctor` or `dtor` is running. In standard C++, virtual calls are shallow during construction and destruction.

Speaking of destructors, a C++/CLI ref class can

have a finaliser as well, to be invoked by the garbage collector, and both the finaliser and destructor must be written so they can be executed multiple times and on objects that have not been fully constructed.

Would you want to explain these things to a class of novice programmers, who yet have only the haziest notion that C and C++ are not the same language, or that Windows is not the only target platform?

We feel that much of the functionality of C++/CLI could have been achieved without doing so much violence to standard syntax. I believe this is valid code for C++/CLI, and merely states explicitly what is now implied:

```
interface class Base {
 public:
  virtual void f1( int i ) = 0;
  virtual int f2() = 0;
};

ref class Derived : public Base
{
  int x;
  public:
  using Base::f1;
  void f1( string s );
  int f2( ) { return x; };
};
```

Apart from the `interface` and `ref` keywords, the same class definitions are valid C++ syntax, and their meaning will not surprise a C++ programmer. To subvert the expectations of millions of trained C++ programmers merely to save typing a few characters now and then (which in any case could be generated by an editor macro or CLI-aware IDE) is unconscionable.

The real damage to C++ and the industry will come not from this or that minor syntax difference or surprising corner case, but from the massive confusion which will be created in the minds of the programming community (and their managers!) over what constitutes valid C++ code.

This confusion is already much in evidence in Microsoft's web pages. The company's online documentation for Visual C++ contains many code examples identified as "C++" – NOT "C++/CLI" or even "C++.Net" – which will fail to compile in a Standard C++ environment. It takes no time at all to find many examples showing parallel code for "C#", "Visual Basic", and "C++" (without qualifier).

An article on "New C++ Language Features" contains this paragraph:

> *The following table lists new keywords that have been added to the C++ language. Note that some keywords consist of two words separated by white space.*

The page goes on to list what are officially context dependent identifiers, but it refers to them baldly as new keywords also, ignoring the subtle difference buried in the draft standard.

Note the statement that keywords have been *added to the C++ language* (no mention that it only applies to this new variant of C++). There is no indication that using any of these new keywords renders code completely non-portable to other environments.

These pages consistently fail to distinguish clearly between Standard C++ syntax and extensions/adaptations for the CLI environment. Microsoft is not the only source of articles in which C++ and C++/CLI are considered equivalent, but they invented this new language and if they cannot tell the difference, it does not create confidence that average programmers will be able to do so.

C++ already has a reputation as a complicated language which is difficult to learn and use correctly. If this incompatible language also called C++ becomes an ISO/IEC standard, it will be perceived that C++ has suddenly become about 50% more complex. The hugely increased intellectual effort would almost certainly result in many programmers abandoning the use of C++ completely.

Future development of the standard language will be damaged, as there will be massive market resistance to adding any additional complexity on top, such as the changes planned for C++0x now undergoing development in WG21.

The C++/CLI draft is not a product of WG21, which maintains the holy document ISO/IEC 14882, but comes from a Technical Group in a different standards organisation called ECMA. A few other companies are involved, but the prime mover in this effort has been Microsoft, with the motivation of supporting its .Net environment. By contrast, joint meetings of WG21 and ANSI J16 (counterpart to the BSI C++ panel) consistently draw participation from 50 or 60 national experts in an international cross section of C++ vendors and customers who work with many different systems and application environments.

If the national standards bodies of ISO/IEC JTC1 should approve C++/CLI as an international standard, it would continue to be maintained by the separate, small ECMA committee and it and Standard C++ will inevitably diverge even more through maintenance. This would compound the already serious damage. The BSI C++ panel will urge that ECMA withdraw its application and if it must re-submit it, find a new name for the language which will not cause confusion.

A parallel to this situation can be found in the history of C++ itself. As related by Bjarne Stroustrup in *The Design and Evolution of C++*, the language in its early days was known as "C with Classes", but he was asked to call it something else: *The reason for the naming was that people had taken to calling C with Classes 'new C,' and then C. This abbreviation led to C being called 'plain C', 'straight C', and 'old C'. The last name, in particular, was considered insulting, so common courtesy and a desire to avoid confusion led me to look for a new name.*

This column should **not** in any way be taken as an allegation that there is a sinister plot by Microsoft to usurp or subvert the standard. Other people may think that, but I do realise that the folks at Microsoft are sincere in what they are trying to accomplish and do have persuasive (to them) reasons why they think these are good ideas. As with their deprecation in VC++8 of standard-conforming but in their opinion "unsafe" code, I think they misunderstand what is important to other people.

Lois Goldthwaite

Website Report

Allan Kelly <allan@allankelly.net>

As some of you may know we have recently suffered a setback on the website redevelopment. I have deliberately not written anything for C Vu about the website for the last couple of issues because the situation was difficult. So here is the story so far.

My first attempt at writing this article was to produce a timeline of what happened. This would have told you what happened and when it happened but would not have given any real understanding and depth. Anyway, much of the project history is already documented in previous issues of C Vu. Instead I think I should discuss the issues with only the basic chronology.

In January 2005 we held an open bidding contest to find a company to redevelop the ACCU website. We called this work Stage 1 and it was be quickly followed by Stage 2. Essentially this stage was to deliver a new server, a Content Management System (CMS) and port our existing content over.

Stage 1 was delivered in May 2005 and at first looked good. But really it was just the old site on new technology with a slightly new look. At the conference in April Tony Barrett-Powell had accepted the position of Web-editor. However it soon became apparent that the tools available in the CMS system were lacking for our needs. At the time we thought we could address this by accelerating Stage 2 but in retrospect I don't think our subcontract ever really appreciated the difficulties we encountered. Maybe we didn't communicate clearly enough.

We issued a specification for Stage 2 and this is where the problems really began. The subcontractor spent an inordinate amount of time before replying to this specification with a tender. In the meantime I took my eye off the ball as I tried to arrange a new database system for the book reviews. And somewhere along the line summer happened and everything slowed down.

When the ACCU decided to redevelop the site and award contracts we were scared of cost. We tried to deal with this through a specification and a fixed-price contract. However we always knew the specification was weak. We hoped that by building piece by piece with extra specs, and fixed-price contracts we could do the job incrementally.

We selected the subcontractor on the basis of price. We chose an organisation that did not understand the ACCU, the values and the way we work.

Our *specification-work-repeat* approach was bad for several reasons. It made more work for us as we wrote the specifications and attempted to make them whole – if not watertight. The process required group discussion and group decision-making. As a voluntary organisation that seldom meets face-to-face it is a difficult and slow process to have such discussions and make such decisions.

As a result our expectations were different to those of supplier. They were good enough to keep working when they started to make a loss on the contract but they didn't tell is that this was so. And that meant we didn't understand their position. Quite naturally our work was put to the back of their queue, we didn't know this and perceived them as responding slowly.

The choice of CMS was wrong too. Four out of five original bidders proposed using a custom CMS, only one proposed using an off-the-shelf (Open Source) system. This proprietary system caused several difficulties.

Firstly the system was light on functionality.

So we found the need to request more features to be added — something neither side had appreciated upfront — and that added to the subcontractor's costs. The system had, from what we could tell, only been used internally by the supplier before we came along. We wanted more control over the system and it didn't support that.

Secondly: the system had not been as widely used, tested and debugged as a COTS system would have been.

Finally, as the project had problems we became more concerned about the propriety nature the system. If we had to move our site to another supplier it would be a big job and, it would only get bigger the more we worked with this system.

By the end of August it was clear there were problems. In September we regrouped and said: we know there have been problems let's try and give it one more go. We set October as our "go live" date and went for it. Our intention was to rebuild our relationship with the supplier over a few weeks and give ourselves the confidence to authorise Stage 2.

October became November but we were close. Then our content went missing, our administration rights disappeared and we received confusing messages from the supplier. The trust we had been working to build up over the last two months was gone.

At this point Tony and myself spoke to Ewan Milne (ACCU chair), some things were clear:

1. We no longer trusted them.
2. If their support and service was good we could live with the poor CMS, conversely, if the CMS system was good we delivered poor service. As it was, neither was good.
3. The ACCU is a voluntary organisation, the time we spend on association activities comes from our free time. We outsourced the website because there was too much work for us to do on this basis. But we found we were spending increasing amounts of our time managing the outsourcing.

Given this, we felt we had no choice but to change. This wasn't an easy decision as it meant writing off close to £4000 of members' money. However, there is no point throwing good money after bad. One should only ever base investment decisions on future expenditure and not on sunk costs, that money was gone.

(The committee briefly discussed action to recover the money but quickly came to the conclusion this was unlikely and would absorb our time, energy and probably cost more money than we would recover.)

I made contact with the company that came second in our bidding process. Coincidentally not only was this the only bidder to suggest a COTS CMS but it was one of only two bidders with ACCU membership. Luckily for us they were still interested, and amazingly, they could started at once.

In mid-November we started over again, this time with Gnomedia and Tim Pushman. Some of you will know Tim from his articles in C Vu and Overload, others may have met him at ACCU conferences.

This time things are looking better. We learnt from the first time and are doing things differently this time around. Specifically:

- We have our website editor, Tony, in place from the start. He is working with Gnomedia on design and content from day one.
- We are working with a subcontract who knows the ACCU and shares our values.
- Gnomedia are working on a time and materials basis. This puts more risk on the ACCU but also gives an incentive to the supplier to be open with those and prompt with the work.
- We are using an existing, off-the-shelf, Open Source CMS called Xaraya. So far this looks impressive and appears to offer functionality out of the box that surpasses the previous system.
- We haven't written a detailed requirements or specification this time but our goal is clear: a replacement website built on a technology that allows for easy updating.

  This may lack preciseness of a requirements document or a rigorous specification but it is something we can all understand very easily. Rather than arguing about details we have a shared vision. This is only possible because we share an understanding and value system with our subcontractor.

We have been working – well Tim and Tony really – actively for about three weeks at the time of writing and things are going well. Communication is much freer, plentiful and productive. Problems are being overcome much easier and faster.

The ACCU is not the first organisation to be caught out like this. In fact we were determined to avoid the mistakes we have seen elsewhere: we knew the problems with fixed-price contracts, we knew the problems the specification is, we knew the problem of outsourcing but we went ahead and made many of them. Simply knowing about things that can go wrong doesn't stop one making the same mistakes.

Of course there are lessons here we can all learn, perhaps even relearn: fixed-price contracts are difficult, outsourcing is difficult, even outsourced projects require some management, avoid the temptation to create your own new technology, I could go on.

We could have chosen not to redevelop the website. We could have given up when the first project failed. We could have navel-gazed and wondered about our failures. Instead we did not do these things, we picked ourselves up and started over again.

If the ACCU cannot produce a new website then we're in danger of becoming irrelevant. If we cannot accept failure we will never try to change anything. What choice do we have really?

I am sorry we spent members money and have nothing to show for it, I wish it could have been otherwise. I give you my personal apology. However I do feel it is better to have tried and failed the never to have tried.

Allan Kelly

The 18th AGM - Announcement

Alan Bellingham
<secretary@accu.org>

Notice is hereby given that the 18th Annual General Meeting of The C Users' Group (UK) publicly known as ACCU will be held during the lunchtime break on Saturday 22rd April 2006 at the The Randolph Hotel, Beaumont Street, Oxford, OX1 2LN, United Kingdom.

Agenda

1. Apologies for absence
2. Minutes of the 17th Annual General Meeting
3. Annual reports of the officers
4. Accounts for the year ending 31st December 2005
5. Appointment of Auditor
6. Election of Officers and Committee
7. Other motions for which notice has been given.
8. Any other Annual General Meeting Business (To be notified to the Chair prior to the commencement of the Meeting).

The attention of attendees under a Corporate Membership is drawn to Rule 7.8 of the Constitution:

> *... Voting by Corporate bodies is limited to a maximum of four individuals from that body. The identities of Corporate voting and non-voting individuals must be made known to the Chair before commencing the business of the Meeting. All individuals present under a Corporate Membership have speaking rights.*

Also, all members should note rules 7.5:

> *Notices of Motion, duly proposed and seconded, must be lodged with the Secretary at least 14 days prior to the General Meeting.*

7.6:

> *Nominations for Officers and Committee members, duly proposed, seconded and accepted, shall be lodged with the Secretary at least 14 days prior to the General Meeting.*

and 7.7:

> *In addition to written nominations for a position, nominations may be taken from the floor at the General Meeting. In the event of there being more nominations than there are positions to fill, candidates shall be elected by simple majority of those Members present and voting. The presiding Member shall have a casting vote.*

As usual, the date and venue is that of the last day of the ACCU Spring Conference. Please note that you do not need to be attending the conference in order to attend the AGM.

(For more information about the conference, including location, please see the web page at `http://accu.org/conference`.)

As far as business is concerned, we are likely to have at least one piece of constitutional business, which will not surprise those who attended last year. On the other hand, since that motion is only intimated and not yet submitted, I cannot actually list it yet.

More details, including any more motions, will be announced in the next issue.

Alan Bellingham
Secretary, ACCU

Dialogue

Student Code Critique Competition 38

Set and collated by Roger Orr

Prizes provided by Blackwells Bookshops & Addison-Wesley

Please note that participation in this competition is open to all members. The title reflects the fact that the code used is normally provided by a student as part of their course work.

This item is part of the Dialogue section of C Vu, which is intended to designate it as an item where reader interaction is particularly important. Readers' comments and criticisms of published entries are always welcome, as are possible samples.

Before We Start

Remember that you can get the current problem set in the ACCU website (`http://www.accu.org/journals/`). This is aimed at people living overseas who get the magazine much later than members in the UK and Europe.

Student Code Critique 37 Entries

Here is a student's attempt at a simple class to provide access to a single database table, `ADDRESS`. Please critique the code and suggest what problems there may be with this class when using it in a larger application, and any issues with the simple test harness.

```
import java.sql.*;
class Scc37 {
  String[] drivers = {
    "sun.jdbc.odbc.JdbcOdbcDriver" };
  String database = "jdbc:odbc:ADDRESS";

  void setDrivers( String[] drivers ) {
    drivers = drivers;
  }

  void setDatabase( String database ) {
    database = database;
  }

  String selectAddress( String query ) {
    try {
      for ( int idx = 0;
            idx != drivers.length; ++idx )
        Class.forName( drivers[ idx ] );
    } catch (Exception e) {
      System.out.println(e);
    }

    String userName="";
    String password="";

    // Get connection

    Connection con = null;
    try {
      con = DriverManager.getConnection(
        database,userName,password);
    } catch(Exception e) {
      System.out.println(e);
    }

    // Execute query
    ResultSet results = null;
    try {
      results =
        con.createStatement().executeQuery(
        "SELECT * FROM ADDRESS WHERE "
        + query );
    } catch (Exception e) {
      System.out.println(e);
    }

    // Get all results
    String retVal = "";

    try {
      ResultSetMetaData rsmd =
        results.getMetaData();
      int numCols = rsmd.getColumnCount();
      int i, rowcount = 0;

      // break it off at 50 rows max
      while (results.next() && rowcount<40) {

        // Loop through each column, getting
        // the data and displaying
        for (i=1; i <= numCols; i++) {
          if (i > 1) retVal = retVal + ",";
          retVal = retVal +
            results.getString(i);
        }
        retVal = retVal + "\n";
        rowcount++;
      }
    } catch (Exception e) {
      System.out.println(e);
    }
    return retVal;
  }

  /** Test harness - 'args' list drivers */
  public static void main( String[] args ) {
    Scc37 scc37 = new Scc37();
    scc37.setDrivers( args );
    System.out.println( "Found:" +
      scc37.selectAddress(
        "name LIKE '%Hardy'" ) );
  }
}
```

From Thomas Hawtin

<tackline@tackline.plus.com>

I will start by covering the obvious errors and move on towards matters of style and how problems of larger applications may be addressed. `setDrivers` and `setDatabase` do nothing - the parameter hides the member variable. The name should be qualified with `this.` to specify the member variable instead of the parameter. To avoid peculiar behaviour, mutable values such as arrays should have a defensive copy made.

```
void setDrivers(String[] drivers) {
  this.drivers = drivers.clone();
}
```

The drivers and database member variables are accessible throughout the package. It is a good idea to limit accessible scope. Variables should almost

always be private. As these variables need to be set only once, they can be initialised from the constructor and made final.

There are many ways of configuring data sources and drivers. For web apps and EJBs, JNDI is the obvious choice. From Java SE 6 standalone applications can have drivers configured within their jar manifest. For test harnesses, system properties make a good choice.

Configuration is a peripheral issue, so I suggest hardwiring for a short piece of code like this. After hardwiring the class is left with no useful state, so there is no need to construct an instance of it. We can make sure no instances of the class are constructed by making it abstract and replacing the default constructor with a private version that always throws an exception.

The exception handling appears to be the minimum to keep the compiler happy. Consider what happens when `Connection.createStatement` really does throw an `SQLException` as promised. The exception is caught and the `results` variable remains as null. The following code will throw a `NullPointerException`. In this case that exception will again be caught and a misleading value returned from the method.

Static exception checking is there to help. Try to be as precise about which exceptions are handled and almost never catch `Exception`. If an exception is thrown, do not just swallow it and carry on. In this short example an exception should be displayed to the developer and the program exit, which can easily be accomplished by declaring `selectAddress` and main methods as throwing `SQLException`.

`Connections`, `Statements` and `ResultSets` are resources which must be closed, whatever the circumstance. The code does not close the resources, even in the happy case. The general form of resource usage is:

```
acquire resource
try {
    use resource
} finally {
    release resource
}
```

The Java syntax pushes programmers towards muddling resource and error handling. Do not be tempted into moving the resource acquisition to within the try block.

The `String` returned from `selectAddress` is created by twice as many concatenations as there values read. Each concatenation allocates and copies the entire string so far. That is an O(n^2) algorithm. With increasing data it will very rapidly become extremely slow. The standard solution is to use a `StringBuilder` and only convert to a `String` at the last moment.

There is a more convenient idiom for building strings with separators than checking the loop index. Initialise a separator variable with an empty string. Assign the separator string within the loop at some point after the append. No condition is required.

There are a few comments that are a substitute for breaking the `selectAddress` into pieces. `// Get connection` marking out a call to `getConnection`, for instance. It is better to split into methods and JavaDoc each. The comment: `// break it off at 50 rows max` is incorrect, as the limit is 40. That is not a helpful comment. More useful would be to make the `while` loop into a very conventional looking `for` loop. The odd feature is the call to `results.next`. In order to prevent potential unnecessary loading of an additional batch of rows, the `&&` operands should be reversed.

The row loop uses the Java, 0-based convention for the index variable. The column loop uses the SQL, 1-based convention. At least be consistent. I prefer to keep to the Java conventions in Java, translating at the last moment.

The variable for the column loop can be moved into the for loop itself. Variable scopes should be as narrow as possible. As we have two loops rather than use `i`, we can match the row variable name with `col` (or column).

If the `selectAddress` method is passed a null argument, then it attempts to execute nonsense SQL. In order for errors to be detected as close to their source as reasonably possible, check that arguments are legal at the start of methods.

`selectAddress` returns may than one record (or none), so should be plural. Also "select" reflects the implementation in SQL. A better prefix is "find".

The class name itself uses initials. The convention for initials is to use the same case for all. It is more readable to use the full expansion – there is no need to be overly terse. If it is reasonable to use a full word, then don't abbreviate it. There is little point in replacing `index` with the less readable `idx`.

`import` statements give an idea as to what types the reader should be looking out for. So `import *` should be avoided unless a large number of types from one particular package are to be used.

It is a good idea to avoid `SELECT *`. If the fields returned change, then your code may well break. You may cause unnecessary work returning fields that are not required. Instead explicitly list precisely what you require.

I am a bit confused as to what the `ADDRESS` table is supposed to represent. What is the meaning of an address name? For the sake of argument, I am going to assume that the table holds customer records. Similarly I assume the database is the customer database.

Dynamically creating SQL is generally discouraged. Failing to correctly escape user supplied strings can allow SQL injection attacks. I have spotted production code very obviously with this problem. The usual solution is to use a `PreparedStatement`, and set the values before execution.

The problem here is identifying what the required functionality is. When it comes to providing an open-ended search interface, that can be a problem. What I have done in my correction is to allow dynamic building of the `where` clause expression, but still use a prepared statement for values. Another interpretation is that the functionality required is to search for a surname, in which case name could be replaced by surname and other fields.

Avoiding copy and paste coding with SQL can be tricky. Move the `ResultSet` dumping code to its own method. It is also straightforward to pass SQL and arguments into the method. Less obvious is how to arrange for the JDBC handling method to call a non-hardwired method to process the `ResultSet`. The way to do that, is to pass in a callback handler object (`ResultSetHandler`). As well as being able to alter member variables of itself, it is useful for the result set handler object to be able to pass back a value through the JDBC handling code.

I have left the JDBC handling code in one big method. In larger applications this would be split. For instance, we would want to keep the connection across multiple statement executions and there are many more things we can do with a `Statement`.

The class contains code working at different levels. As it grows, that should naturally split. In particular the test code can go in a separate source directory from the production code.

An Object-Relational Mapping tool, such as Hibernate or an EJB3 persistence implementation, would get out most of the repetitive JDBC and SQL.

In order to save space, I have indented only two spaces and elided the JavaDocs.

```
import java.sql.ResultSet;
import java.sql.SQLException;

interface ResultSetHandler<RESULT> {
  RESULT resultSet(
    ResultSet results
  ) throws SQLException;
}

abstract class StudentCodeCritique37 {
  private void StudentCodeCritique37()
   throws Throwable {
    throw new Error();
  }

  private static String
  findCustomersWithNameEnding(
    String nameEnding
  ) throws SQLException {
    if (nameEnding == null) {
      throw new NullPointerException();
    }

    return findCustomers(
```

```
    "name LIKE ?", '%'+nameEnding
  );
}
private static String findCustomers(
  String whereExpression, Object... args
) throws SQLException {
  if (whereExpression == null) {
    throw new NullPointerException();
  }
  String customerFields =
    "name, street, town, county, postcode";

  return executeQuery(
    new Querier<String>() {
      public String resultSet(
        ResultSet results
      ) throws SQLException {
        return resultSetToString(results,
         40);
      }
    },
    (
      "SELECT "+customerFields+
      " FROM customer WHERE
       "+whereExpression
    ),
    args
  );
}

static <RESULT> RESULT executeQuery(
  ResultSetHandler<RESULT> handler,
  String sql,
  Object... args
) throws SQLException {
  if (sql == null || args == null) {
    throw new NullPointerException();
  }

  java.sql.Driver driver =
    new sun.jdbc.odbc.JdbcOdbcDriver();
  java.sql.Connection connection =
    driver.connect(
      "jdbc:odbc:CustomerDB",
      new java.util.Properties()
    );
  try {
    java.sql.PreparedStatement statement =
      connection.prepareStatement(sql);
    try {
      int argNum = args.length;
      for (
        int argCt=0; argCt<argNum; ++argCt
      ) {
        statement.setObject(
          1+argCt, args[argCt]
        );
      }
      ResultSet results =
        statement.executeQuery();
      try {
        return handler.resultSet(results);
      } finally {
        results.close();
      }
    } finally {
      statement.close();
    }
  } finally {
    connection.close();
  }
}

private static String resultSetToString(
    ResultSet results, int rowLimit
  ) throws SQLException {
    if (rowLimit < 0) {
      throw new IllegalArgumentException();
    }

    java.sql.ResultSetMetaData metaData =
      results.getMetaData();
    int colNum = metaData.getColumnCount();

    StringBuilder buff = new StringBuilder();
    for (
      int row=0;
      row<rowLimit && results.next();
      ++row
    ) {
      String sep = "";
      for (int col=0; col<colNum; ++col) {
        buff.append(sep);
        buff.append(results.getString
         (1+col));
        sep = ",";
      }
      buff.append('\n');
    }
    return buff.toString();
  }

  public static void main(
    String[] args
  ) throws SQLException {
    System.out.println(
      "Found:\n" +
      findCustomersWithNameEnding("Hardy")
    );
  }
}
```

From Jim Hague

< jim.hague@acm.org>

The preamble to this problem begins by describing the problem as a Java problem. I think this understates the problem. The task requires Java to interact with a database via the standard basic Java mechanism for working with SQL database, JDBC. JDBC deals with the nitty-gritty of how to connect to a database, send SQL commands to the database, receive the results, and make use of them from within Java. It requires you not only to know at least some SQL as well as Java.

Often, of course, it also requires you to be able to configure the entire environment to test the program. You have to be able to identify and install the correct JDBC drivers and to configure the database to authenticate and accept commands from JDBC programs. Assuming the code sample compiles, runs and produces at least one address (I can't test it right now), the student has surmounted several hurdles and produced what might be called a 'Hello, world' JDBC program. Kernighan and Ritchie noted in *The C Programming Language* that successfully compiling and executing a program requires knowing how to work the local compiler and linker, a task where the author of a book can provide steps for specific systems but can't hope to cover all extant systems. Connecting to database adds an extra layer to the problem. The student is due some credit for overcoming those hurdles.

I've not been working on Java code daily for while now, and on a first glance my impression was relief. It's easy to follow what the code is doing, and there are even a few comments to help you on your way. It would be nice if each method had a proper Javadoc comment, and some folk deplore wildcards in imports. But something else nags...

Ah! There are no access specifiers. Not on the class, not on the member variables, or the member functions. Java has 4 access specifiers: `private`, `protected`, `public` and default of 'package private'. This isn't the place for a full description of access specifiers; I'll just say that judging by the code the intention is that the class should be generally available, and that the `setDrivers`, `setDatabases` and `selectAddress` methods should be generally

callable. In that case, they should each have their declarations prefixed `public`.

```
public class Scc37 {
    // ...
    public void setDrivers ...
    public void setDatabase ...
    public String selectAddress ...
}
```

In larger programs, classes are further categorised into packages. When the student is finished reading up on access specifiers, they should turn to coverage of packages.

The next thing I noticed was the error handling. Inside the main `selectAddress()` method, each step along the way is wrapped in a `try..catch` block and if any exception occurs it is reported by printing it. Execution then proceeds to the next section of code, which is almost certain to fail too as a consequence of the previous failure. Meanwhile, the caller of `selectAddress()` never sees any of these exceptions, and is unable to tell whether getting an empty result is due to errors (e.g. can't load driver/access database, invalid SQL) or simply because there are no records in the table which match the selection criterion.

What to do when errors strike is never an easy question, and it's one of the big things that separates 'Hello, world' programs from Real Code. In this case, my feeling is that the `selectAddress()` method simply can't have any knowledge about what would be an appropriate response to an error. Should it exit? Should it retry the operation? In any event, the calling code has to be told when an error occurs rather than no data is found. My initial inclination is to chuck the whole problem upstairs by not fielding exceptions at all. Change the declaration to indicate that an `SQLException` may be thrown, and remove all the `try...catch` blocks from the code.

```
public String selectAddress(String query)
    throws SQLException
{
...
}
```

While on the subject of things better handled by the calling code, I next want to talk about the first two operations `selectAddress()`. It first loads the JDBC driver, and then requests a connection to the database. The connection is assumed not to require username/password authentication.

Opening a connection to a database is a heavyweight operation when compared to doing a simple `SELECT`. Connections are relatively precious resources, and your application performance – and quite possibly you if your database admin catches you – will suffer if you open a connection on every `SELECT`. In the 'Hello, world' case it doesn't matter; we're only doing the one query. Larger programs often keep a pool of connections, doling a connection out to code that needs one and receiving it back when that code has finished its business with the database, but keeping the individual connections open as long as the application needs them. So here a class intended for a larger application should rely on the application to supply the connection. That in turn means that it is the application's responsibility to load the JDBC drivers and worry about authentication, so the class now shrinks to just the one method.

```
public class Scc37 {
    public String selectAddress(Connection
con, String query)
        throws SQLException
    {
    ...
    }
}
```

Having removed about a third of the lines in `selectAddress()`, I now want to make a few points about the remaining code that executes the query and makes up the result.

The first point is that the exact results obtained are very dependent on the table declaration in the database. I think SQL will return the columns in the order in which they are declared in the table if requested to `SELECT *`. This may be fine; the requirement for `selectAddress()` may indeed be just that it returns all information in `ADDRESS` comma-separated in a string. More usually, though, calling code is after specific bits of information, like the town or postcode. In that case, you should either specify the order in which columns should be returned (e.g. `SELECT NAME, ADDRESS1, ADDRESS2, TOWN, POSTCODE`), or use the alternate version of `ResultSet.getString()` which takes the name of the column to return (for example, `retVal = results.getString("POSTCODE")`). From memory the former is slightly more efficient.

The second point is the code that returns data only for the first 40 rows found. In general, if you find yourself writing any code that discards data returned from a SQL query, that is the time to learn a bit more SQL to see if the SQL query could have been formulated to omit that data. In this case, consider what will happen if the query runs against an address table and matches 10 million entries. The database will dutifully pick out and return the data from 10 million records. This isn't going to happen quickly and if as is usually the case the database is on a different machine you're going to be schlepping a lot of data across the network unnecessarily. I'm no SQL guru, so I'll just mention that on MySQL at any rate the `LIMIT` command achieves what we want here. Oh, and one more thing here. Why set the cap at 40 rows? Calling code may want 100, or it may want one.

Finally, I note that the documentation for `ResultSet.getString()` says it returns 'null' if the column contains SQL NULL. This 'null' will cause trouble when building `retVal`. You need either to ensure that the column can't contain NULL, or look out for and handle getting 'null'.

We're now left with a class with one method, a `main()` test method and no member variables. Furthermore, the code calling the `selectAddress()` routine must supply a query parameter that is a legitimate `WHERE` clause for the main query, and as the test shows that clause is going to have to know about the table structure. So, all in all, the class isn't buying us a great deal; Java code has to know about the underlying table because of that `WHERE` clause, and can't pick specific bits of address information out of the result. And this brings me to a fundamental design point. The class doesn't abstract addresses. It's a small wrapper around querying a table. I feel that in any real-world application the design would be better separating out addresses from their handling. For a UK-centric application something like:

```
public class Address
{
    public String getName();
    public String getAddress1();
    public String getAddress2();
    public String getTown();
    public String getCounty();
    public String getPostcode();
};

public class AddressTable
{
    public Address[] getByNames(String name,
       int first, int max);
    public Address[] getByNamesLike(String
       pattern, int first, int max);
}
```

Finally, a quick word on testing, the student has included a simple test routine in a `main()` routine. Good. It's a start. Unfortunately, there is no way apart from examining the program output to determine if the test has passed or failed. Fine for a small one-off, but not so good if we're testing a lot of queries – will you always spot the error output? And fatal if we want to run the test in an automated suite. At a mimimum I would recommend giving an appropriate `System.exit()` value. From there the student can proceed to a study of JUnit and similar.

Commentary

I thought it would be interesting to break with tradition and use some Java sample code, but this didn't actually seem to produce any more critiques than usual – unless it was simply everyone was away on holiday.

I can't believe so few members of ACCU are competent to write code critiques, so I thought I'd use some of the commentary to talk *about* the process of writing a critique and hopefully encourage some more SCC entrants in the future.

What can entering the Student Code Critique do for you? The first thing is that you move from being a passive consumer of the magazine to actively working with the example code. This is a good skill to practice whenever you are reading (and not just in IT!) – being actively engaged in the subject matter helps you to learn more and avoid unquestioning acceptance of what may be incorrect or less than ideal.

By engaging in looking for problems in the published code you can hopefully improve your skill at reviewing code – which is an important component of many programming methodologies. (However, don't think this only applies to the student code critique column, you can be thinking critically about all the code you read in the computer press. Just because you see the code in print does not mean it has no faults!)

Then you must try to turn your critique into prose that can 'stand on its own' without you being present to explain what you meant to the magazine's readers. As those of you who try this have probably found, it can be hard at first but again it is a good skill to seek to improve.

The beauty of a competition is the best entry (or sometimes the only entry…) wins the prize, so you don't necessarily need to produce the 'perfect' entry, even should such a thing exist. However we also hope that some of those who write a critique might then extend their writing further to produce an article for C Vu or Overload.

The prize – ah yes – a free book. How many of us read as much as we should about IT? [Apart from Francis, perhaps] If you choose well your book should teach you something of value that will broaden your knowledge of this ever-increasing world of computing.

So, why not give it some thought this issue if you've never before attempted an entry for the student code critique?

To return to the Java sample code, the critiques above have covered most of the problems with the code.

The basic syntax problem where a member variable has the same name as an argument is of course relatively obvious in very short methods like this example – but can be hard to track down in larger methods. The problem of arguments matching member variables can sometimes be detected at compile time by use of `final` on the arguments – this will at least catch attempts to assign to the variable where assignment to the member variable was expected. It always puzzles me slightly that Java allows local variables and arguments to hide member variables but doesn't let you nest variable names inside scopes:

```
int aValue = 1;
{
  int aValue = 2; // OK in C++, not OK in Java
}
```

The only additional problem with exception handling in the code that I'll mention is with the `for` loop in `selectAddress`. Since the exception handler is *outside* the for loop it means that any exception will terminate the loop early. This sort of bug can be hard to detect.

The final issue – that of configuring the database sources – seems to be getting more complicated with each release of Java. Since it depends on the environment in which your code is running (stand-alone, in a Web server, etc) the key item is to try and localise any code with any knowledge of this configuration – Jim's solution of passing in the database connection is an example of this approach.

The Winner of Student Code Critique Competition 37

The editor's choice was very hard to make as both entrants where very clear in their approach and where the problem was. I hereby wave my editorial wand and say that both deserve the prize on condition that they get a review of it to me for a future edition of C Vu.

Please email `francis@robinton.demon.co.uk` to arrange for your prize.

Student Code Critique 38

(Submissions to `scc@accu.org` by Mar 1st)

The student wrote "I'm getting a compilation error with this program, something about the instantiation of `invArg;` I think the type checking is too strong; any suggestions?"

```
#include <iostream>
#include <cstdlib>
using namespace std;

template<class T>
class invArg
{
public:
invArg(T& arg):inv(arg){}
virtual void Write()const{cout << inv
 << endl;}
private:
T inv;
};

template<class T>
class Exp
{
public:
T operator()(const T& base,
 T exp)throw(invArg<T>)
{
if(exp<0)
1/operator()(base,exp);
else if(base==0)
throw invArg<T>(base);
else
{
T ret=1;
for(;exp--;)
base*=exp;
return ret;
}
}
};

int main()
{
for(;;)
{
try
{
long double base,exp;
cout << "Enter a base and an exponent: "
 << endl;
cin >> base >> exp;
cout << base <<"^" << exp << "=" << fixed
 << Exp<long
double>()(base,exp) << endl;
}
catch(invArg<long double>& inv)
{
inv.Write();
}
system("PAUSE");
return 0;
}
}
```

Good luck!

Letters to the Editor

There is a line in one of the Christmas songs which goes “well, I wish it could be Christmas everyday”. While I'm not advocating such an activity (not unless I get a healthy pay rise and have a guaranteed 360 days off a year), I must say that the Yuletide has brought in more letters (well, emails) to me than since I took over 2 years back.

Book Reviews

Quite a few have expressed that they think the idea of not putting the prices of books next to the books a good idea. The consensus is that because of the variance of prices between traditional booksellers and the likes of Amazon, it seems pointless to have the prices printed on paper or online as they are pretty much useless, even as a guide.

There has also been some disquiet over the swap over after the sad loss of Chris last year, the majority being the slow update on the website. As you've seen in the Officer's reports section, Allan Kelly has given an insight into the problems we are currently experiencing with the site. Be assured though, everything is being done to try and solve this problem.

Finally on books, early last year a review was set in motion about the whole book process. This has been restarted and I should (hopefully) have something to report in the next issue.

Debuggers

The piece on Undo seems to have caused the following from Terje Slettebo

Debugging is obsolete!

If you find yourself doing a lot of debugging, you're probably doing something wrong. The reason for the lack for interest in debugging I think has much to do with the increased interest in agile development, including the practice of Test-Driven Development. The following article is a good introduction to that (http://www.phpbuilder.com/columns/baker20040202.php3):

"After two years of coding this way, if I am ever in that interview I will give a different answer now. "Test, code then design" If they ask me what happened to the debug phase I have only one answer... "What's debugging?" "

As Robert C. Martin says in his blog at Artima: "Debuggers are a wasteful Timesink" (http://www.artima.com/weblogs/viewpost.jsp?thread=23476).

If you have to single-step through the code, either manually or with the help of a debugger, to understand what a program does, rather than being able to understand it by looking at the code, then it's a sign that the code is not as clear as it should be, and/or you have insufficient tests. The solution is then to refactor, and/or add tests, not single-step.

I've found over the years that I spend less and less time debugging, and more time on "massaging code into shape" (refactoring), doing continuous refactoring/design. Typically, when you clean up code, what used to be a hidden (maybe even latent) bug, tends to jump out at you, when the code becomes clearer.

Maybe the real "debugging scandal" is that way too much time is spent doing debugging, rather than properly designing programs (such as with refactoring). As "Uncle Bob" says in his above mentioned blog:

*"And yet, for all their power, debuggers have done more to damage software development than help it. [...] IMHO a debugger is a tool of last resort. Once you have exhausted every other avenue of diagnosis, and have given very careful thought to just rewriting the offending code, *then* you may need a debugger."*

In order to address these points, I asked Greg Law of Undo.

I have heard the "debuggers considered harmful" argument several times, and it will not surprise you to hear that I do not agree with this position.

In an ideal world, it is of course better to write bug-free programs than poorly crafted code that is debugged later. Alas, real world constraints often make this impossible. Here are just a few examples:

1.Bad assumptions. If you as the programmer have a misapprehension about your program's environment (e.g. a library routine's behaviour), then no amount of thorough design will prevent your bug. Note, I'm not talking about when you realise that you don't understand the environment. It's when you think you understand it but are mistaken that things get tricky. In other words, it is by definition impossible to foresee unforeseen problems!

2.Someone else's code. If you've never been asked by your boss to fix a bug in someone else's code, then you're in a very small minority. Likewise, if rewriting that shoddy code-base is a viable alternative, you're also lucky. Any tools that can help here are extremely valuable.

3.Difficult or poorly defined problems. Often you don't understand the true nature of the problems you need to solve until you've already written some of the code. The waterfall development model looks great on paper, but we all know it works in practice only for the simplest of projects.

I would however agree with the statement "debugging should be the last resort" (as opposed to "a debugger should be the last resort"). It is always desirable to eliminate bugs as early as possible, and so much the better if this can be at the design stage. But sadly, all too often code does need to be debugged. Once we arrive at this sorry state, we need all the help we can get. This means refactored code, better tests and, yes, tools such as debuggers.

SCC

It's not often that I hear anything about the SCC (except when Roger emails me the submission). However, null references seems to have triggered this from Fazl Rahman.

Can we stop talking about null references in the context of c++ please.. Thank you.

In the Dec 2005 C Vu, Jim Hyslop (talking about "null references" on p.7) recommends "Do your best not to accidentally create them, and never, ever deliberately create them." This puzzled me as my mental model of a C++ reference is a 'pointer that automatically dereferences at each use'. In this model, one cannot create a null reference, deliberately or accidentally, as it requires dereferencing a null pointer.

Here is a thread from comp.std.c++ in Google on this topic: `http://tinyurl.com/bzhfx` *with the likes of Pete Becker discussing this, where the following appears:*

The Standard addresses this in 8.3.2/4, which includes

Note: in particular, a null reference cannot exist in a well-defined program, because the only way to create such a reference would be to bind it to the "object" obtained by derefencing a null pointer, which causes undefined behavior.

I checked this on my copy, found it and asked Jim..

*Keep in mind the key phrase in that quote from the standard: "in a well-defined program." This does not mean that it is impossible to create a null reference, only that it is impossible to create a null reference *in a well-defined program.* While this may seem to be splitting hairs, it is an important distinction: your compiler cannot always detect when a program is ill-defined.*

Consider this program:

```
// file1.cpp
void g(int&);
void f( int * ptr )
{
  g( *ptr );
}

// file2.cpp
int main()
{
  f(0);
}
```

This program creates a null reference, and so is ill-formed. But I'm willing to bet no compiler currently available will diagnose the error, so the compiler will happily compile and link the program. Even if a compiler could diagnose that error, it would be almost impossible to diagnose if f() is passed a pointer whose value is set at run-time, which may or may not be null.

So until and unless the language evolves to the point where it is impossible to create them, I'm afraid we can not stop talking about null references in C++, and my advice stands. C++ programmers must be aware of the issues, and know how to avoid falling into that trap.

And yes, there *are* programmers who believe that creating null references is OK: for every person like the one who asked the question in comp.std.c++ there is at least one programmer who says "It works, therefore it must be OK" (or worse, "COM does it, so it must be OK").

That's all the postbag for this edition. Please remember, the dialogue section of C Vu is as much for you as it is for the regular contributors and committee. Have you voice heard. Email cvu@accu.org if you have anything to say about the magazine, contents or what you would like to see in future editions.

Francis' Scribbles

by Francis Glassborow <francis@robinton.demon.co.uk>

Thoughts for a New Year

I am writing this on the first day of 2006. I started to think about how the world of computers and programming has changed over the years and the ways in which it has not.

The computer I am writing this article is more powerful than the most powerful machines of the mid 1970s. Since then I have become comfortable with using a keyboard to do my writing, and regularly turn out printed text that is better presented than much of the professionally laid out material of that time.

Computers have changed the way that most of us write, but how much have those teaching writing skills changed their teaching? I suspect the answer is 'not much.' I think that those changes that have happened are often for the worse. Word-processors should allow students to focus on the process of writing rather than the mechanics of doing so. Do our children's teachers require that change of focus or are they still ploughing along with the old ways dressed up in some superficial new clothes? For example, are they still focused on rules for presentation that were designed for hand-written or typed text?

Most people still consider that computers are mathematical tools; so why is it that much maths being taught today is essentially uninspiring and taught by uninspired teachers? Yes, there are many good teachers in the classrooms of the world but too many of them are having their work destroyed by curricula that offer the pupil nothing.

Computers are magnificent tools for both creative writing and mathematical investigation but it is much easier to teach routine material which can be assessed by some objective criteria. Teaching has not got better over the last fifty years, the apparent improvement in the examination results is, in my opinion, almost entirely due to the revamping of examinations so that it is possible to train pupils to get good grades. Teaching that way may produce excellent performance assessments for the teacher and the school, but such teaching is stultifying and brain numbing.

Most of us understand that giving others the tools and knowledge to feed themselves is much more effective in the end than giving them food. So why are we so poor at feeding young minds?

As is often the case when I sit down to write, I find myself writing something very different from what I had planned. I intended to write about how programming has failed to change with the hardware.

Many programmers are still locked into tools that are 25 or more years old in concept and design. For example, most programmers use mono-spaced fonts and text editors that use simple ASCII coding. Mono-spaced fonts were a combination of what teletypes produced and the need to keep track of the columns (for example a traditional punched card had 80 columns; in FORTRAN IV columns 1-6 were for labels and columns 73-80 were for the card sequence number – useful when you dropped your card deck). I have heard a variety of arguments put forward for continuing with mono-spaced fonts. The strongest being that all the third party tools for processing source code require files that are in a simple format. With respect, even that is plain stupid. It is easy to make a word-processor that generates parallel files with text in one and format in the other. Human beings can then have the source code presented for easy reading, and the development tools can have the simple text that they need.

This reminds me of the struggle I first had with a keyboard. In my early days of programming, I wrote the first draft of my source code in long hand, and then keyed it in to some computer readable form. Eventually I found that I could dispense with the hand-written stage. (These days the problem is learning to use speech recognition methods. I may eventually learn to dictate my source code to my computer.)

If the only problem with programming was the use of unimaginative tools I suppose I could shrug my shoulders and put up with it, but the more fundamental problem is in the programming languages we are using. They are inherently designed to support software development for a machine with a single processor. There has been hardly any development of language features for computers that have more than one processor (or core in a multi-core CPU) running at a time (yes, I do know about languages such as OCCAM). I have listened with increasing dismay to the explanations of others about how clever compilers, instruction schedulers etc. mean that human beings do not need to worry their little minds with parallel execution of code.

Those responsible for languages like C and C++ are honest enough to admit that those languages are based on the assumption of sequential execution. The problem is that this assumption is buried very deep in computing. The tools for multi-threaded programs still assume that there is just one processor that switches between threads. In other words if one thing is being done, nothing else will be executed at the same time.

However the newer languages such as Java, C# and Python have the same assumption, it just has not been so clearly stated. The popular programming languages assume that things like multi-threading are done by a form of task switching. When that is not the case because different threads are running on different cores (and if they are not your program is not making effective use of the hardware) you have serious problems waiting to manifest at the worst possible time.

The assumption that only one instruction is executed at a time is not true any longer. I was recently talking with some of Symbian's (responsible for the Symbian OS used on many mobile phones) staff when the subject of the C++ abstract machine came up. What shocked me was to learn that the next generation of mobile phones will use multi-core processors. At least that company was aware that this change had fundamental impact on their programming methods.

Software is steadily taking over the world. That means that getting it wrong has an ever increasing potential for disaster. Your wireless, your TV, your MP3 player etc. are all essentially software supported by some hardware.

The world has changed in ways that many have not noticed. Most people see the outward changes but completely miss the changes in how the task is carried out. If programmers cannot adapt to the changing world of software and silicon what hope has the rest of the human race?

Global warming may create a disastrous climate change but I suspect that our inability to handle our changing technology is just as big a threat. We need to break away form our comfortable belief that it will all work out in the end. Only when we accept that we have a problem and do not know what we are doing will we be in a position to do something about it.

Commentary on Problem 23/24

I am republishing this problem because no one responded. I know that some readers have an idea as to what the problems are. I also have an SQL problem which I will publish next time.

Problems with initialisation have been of concern to those responsible for working on the next version of the C++ Standard. Have a look at the following code and comment on any possible surprises.

```
#include <iostream>
struct X {
   int i;
   X(){}
};
struct Y: struct X{
   int j;
   Y(): X(), j() {}
};
Y y = Y();
int main(){
   std::cout << y.i << std::endl;
   return 1;
}
```

Please focus on the interaction of an explicit constructor that does not initialise all the member data and the rules for global initialisation.

The second problem with the code is that while the Standard guarantees the result of returning zero from `main()`, it does not guarantee what will happen if you return any other integer value.

Cryptic clues for numbers

Last issue's clue

Help! Looks like a sailing dinghy. Hawaiian police series number 5. (3 digits)
The answer is 505 (I have not had any entries, so that saved me finding a prize.)

This issue's clue

On reflection, this issue is still the same prime. (3 digits)

If you wish to take part in this competition, please remember that you have to supply an alternate question for the same answer before you can claim the prize!

Francis Glassborow

Francis Glassborow is a freelance computer consultant and long-term member of BSI language panels for C, C++ and more recently Java and C#. He is a regular member of the UK's delegations to WG14 and WG21. He is also the author of 'You Can Do It!' and "Introduction to programming for novices".

Features

We Own All Your Computers...

Alan Lenton <alan@ibgames.com>

Media Police! FREEZE! Move your hands slowly away from the keyboard, keeping them visible at all times. Do not attempt to touch your mouse...

It all started when Mark Russinovich ran a series of regression tests on a security program he was working on. To his amazement the program reported that a rootkit was installed on the computer.

A rootkit is a program designed to hide the existence of other programs from the owner of the computer. The main users of rootkits are virus, spyware, and other malware writers – and, of course, malicious hackers covering their tracks.

Eventually, after some sleuthing Russinovich discovered that the rootkit had been installed by a SonyBMG CD – VanZant's 'Get With The Man'. In this case the purpose of the rootkit was to hide the fact that the offending CD had changed the configuration of the operating system (Windows) by installing a new device driver. The CD was using a copy protection product called XCP written by a British company called First4Internet.

Most of the tools used by hackers have legitimate uses as well as malicious ones. Rootkits, though, are one of the few programs that don't fall into this category. There are no legitimate uses for a rootkit. Once a rootkit is installed, anyone who knows about it can use it to hide things from the computer's owner. You can easily imagine the alarm this news generated when Russinovich reported his findings in his blog!

But finding the rootkit was only the start. When he tried to remove the driver he discovered that Sony had failed to provide an uninstaller, and that when the driver was removed manually, the computer stopped being able to access the CD drive. Worse, an examination of the driver code revealed that removing the driver could completely crash the computer.

Sony's response to the revelation that they had been illicitly tampering with their customers' computers was summed up by a comment from Thomas Hesse, President, Global Digital Business, Sony BMG Music Entertainment: *Most people, I think, do not even know what a rootkit is, so why should they care about it?*

To make things worse for Sony, it was then discovered that the software was contacting Sony when you played the CD. Sony promptly issued a denial that there was any code to contact them. This was followed by an equally prompt number of posts on the web containing detailed transcripts of the software communicating with a Sony server. Oops!

At this stage Sony backtracked and admitted that the software did indeed phone home, but that it was only to see if there was any updated artwork to display. I'd guess that was probably true, but by now, who was going to believe anything Sony told them?

Needless to say, the sounds of class action lawyers sharpening their knives could be heard throughout the internet, though Sony seemed to be oblivious to the noise.

By now the news of Sony's rootkit had spread far and wide and hackers were starting to sit up and take notice. First off the blocks were World of Warcraft Online hackers who developed a version of their cheat software that used the rootkit to hide their cheats from World of Warcraft's security program.

Under attack from all sides – the story had by now reached the mainstream media – Sony produced an uninstall program to remove the rootkit, but not the driver itself. If you wanted to remove the driver you had to go to the Sony web site, jump through hoops and provide Sony with all sorts of information it had no right to ask for.

Within another day or so the first viruses using the rootkit had appeared, and Sony belatedly began to realise that they couldn't just wait for this to go away. They announced that they had halted production of music CDs with this particular protection, and that they would *re-examine all aspects of our content protection initiative*. Even so, they still didn't plan to recall all the CDs with the rootkit on.

By now a number of interesting sub-themes had emerged from this fiasco. For instance at one stage Sony was trying to claim that users had agreed to having the rootkit installed because it was in the end user license agreement (EULA). This caused a lot of argument about whether that was the case or not, though my feeling is that it wasn't. That debate, however, completely missed the main issue – since when have you had to agree to an end user license to listen to a music CD?

Another other sub-theme that ran through the discussion – particularly in the blogs – was that of a conspiracy to seize control of people's computers by the big media companies. I'm sure big media would love to do that, but I don't think this was such an attempt. Never impute malice when greed and incompetence is sufficient!

One little covered aspect of the affair was that you could defeat the copy protection by putting gaffer tape (duct tape for my US readers) onto the outer edge of the disk. I don't really recommend you do that, since disabling the autorun facility of the computer's CD drive will also stop the rootkit and DRM from autoloading, after which you can play the music with any software you choose.

And finally, of course, those who bought legitimate CDs drew lessons from this affair that were very, very, worrying to the big media companies. The legitimate owners of the music had their computers made more vulnerable to malware and had made the operating system less stable. Those who obtained pirate copies over the net and didn't pay anything suffered no problems, and were OK.

All this would have been bad enough, but the following week further revelations completely undermined all Sony's efforts at damage control.

A number of people had been looking at Sony's code a little more closely, and what do you think they discovered? Nothing less than code taken from an open source MP3-encoder called LAME – in violation of its LGPL copyright license. And all this in the name of enforcing Sony's own copyright...

By now even Microsoft had noticed what was going on. They let it be known that they had decided to classify the XCP copy protection system as spyware, since it met the 'objective criteria' Microsoft uses to assess potentially malicious programs. Being on the side of the good guys must have been quite a novel experience for Microsoft!

While all this was going on Sony came under attack from a completely different direction – allegations of Internet 'price-rigging'. It emerged that Sony and other manufacturers have been accused of asking online retailers for 10-15 per cent more for wholesale electronic goods than they charge their bricks and mortar counterparts! Sony is already facing investigations by the UK's Office of Fair Trading (OFT) and the European Commission over its pricing strategy.

Meanwhile, back at the ranch, Sony finally bowed to consumer pressure and agreed to withdraw and replace all the CDs affected by the rootkit. According to Sony about 4 million copies had been manufactured, and some 2.1 million sold.

But What of the Compromised Computers?

Well, Sony eventually issued an 'uninstaller'.

Hooray!

Errrrrr, no actually. Security researchers soon discovered that the cure was even worse the disease. In this case the cure took the form of an Active X control installed via Internet Explorer. Unfortunately, the settings in the control will cause viewing a maliciously crafted web site to compromise the viewing computer.

Those of you holding stock in First4Internet, purveyors of fine rootkits to media giants, may like to consider disposing of it as rapidly as possible :)

By Christmas, Sony was facing a slew of class-action suits, a possible action in Italy, and an action from the State of Texas. And to cap it all, one of its other copy protection programs, MediaMax, was also under the microscope and revealing its own set of problems.

Eventually, in the hiatus between Christmas and the New Year details of the class action settlement in New York State slipped out. It was pretty harsh, and deservedly so. There was compensation for affected buyers of

[concluded at foot of next page]

Silas' Corner

Silas S. Brown <ssb22@cam.ac.uk>

Putting Old Modems to Use

Most technical people are moving to broadband or faster Internet connections, and may be throwing away their old dial-up modems. But if you still have a landline then you may occasionally find that old modem useful apart from giving you a slow Internet connection.

Fax

This is perhaps the most obvious application, since so many modems are sold as "Fax Modems" with the ability to send and receive faxes. Using the modem to send a fax will save printing a hard copy and may also save purchasing a fax machine, especially if you need to send faxes only occasionally (email will do nearly all the time nowadays). Sending a fax from your modem may also be useful if for some reason you need to send a fax that's so sensitive you don't even want your own receptionist to see it (but of course I'm not suggesting the theft of company resources).

Modems usually come equipped with DOS or Windows software to do faxing. Of the Linux software that's available, perhaps the simplest to set up is "efax"; alternatives include "hylafax" and "sendfax". If you are sending internationally then you may wish to use one of the many companies that provide discounts, in which case you will need to instruct the software to dial their numbers first and to pause for long enough.

SMS

This may be a very useful application for some organisations. Using the SMS (Short Message Service) on mobile phone networks to send "text messages" is possible from just about any modem. The software needs to dial the number of a mobile company's SMS message centre and to speak its protocol. This call will cost something, but it's not much more than a short call to a mobile phone, and if you have several messages to send then you can often combine them in one call. The Linux package "smsclient" seems to be the most reliable approach; there are also various Windows programs that claim to be able to do it.

Textphone

The UK Textphone service is a text-based telephony service for people who can't use voice telephones (usually because they are deaf). Many organisations have a textphone number. If you have deaf friends, or you are an organisation that wants to be reachable by textphone, or if you temporarily lose your hearing and need to talk to a doctor about it, then you may wish to use this protocol. If you have a permanent need then you should be able to get a real textphone, which is a special kind of telephone that has a built-in keyboard and display. But for temporary or occasional use, you can use your modem. A textphone is simply a 300-baud V21 modem (yes, 300 baud; these things have a long history), so if you set your communications program to enforce 300 baud and to give you local echo (since the remote side will not echo the characters you type), you will be all set to make and answer textphone calls directly with the other caller's unit (there are no message centres or other intermediary servers in the textphone protocol). Textphones have only 1-line displays so carriage returns are meaningless; if your software has a word wrap option then enable it. If your software asks which terminal to emulate then set it to something simple such as VT100.

Please do not use the textphone protocol just because you fancy having an IRC-like chat with an organisation instead of calling them. Organisations with textphone numbers tend to have far more voice operators than they have text operators, and there might be someone with a genuine need for the text line that you're on. You should use this protocol only if you or the other person (or both) cannot use voice. You also need to be familiar with the common abbreviations; the most important one is GA for Go Ahead (equivalent to "over" in 2-way radio).

Other Uses

If you share a modem with a voice line then you could use the modem to dial your voice calls if there's some reason why you can't do so yourself. Just remember to tell the modem to disconnect as soon as it has dialled. You could also get your modem to page you on your mobile (ring it a couple of times) to indicate that some significant event has happened, although a text message may cause less confusion.

And then there's always using the modem for a dial-up Internet connection. Many rural areas are still not covered by broadband, so if you yourself don't have a need for the modem any more then you may well know someone who does.

Silas Brown

[continued from previous page]

Sony CDs, software utilities to remove the offending copy protection, the recall of the XCP CDs and no manufacture of MediaMax CDs for at least two years. On top of this there were also a series of other measures agreeing not to collect personal information, and a waiver of rights obtained through the use of the EULA .

I'm assured that rumours that Mr Hesse, quoted earlier, is retiring to spend more time with his rootkits, are totally unfounded.

But the real unanswered question from this fiasco is one that the mainstream, and even the technical, press have been noticeably reluctant to ask. Why didn't the Anti-Virus companies spot this rootkit? It's not as though rootkits are some sort of never before seen new-fangled attack – they are old hat in the security world. Where were Symantic, McAfee, Computer Associates and their ilk while Sony was installing its rootkit?

You can take your choice of answer here – incompetence or collusion with Sony. If just one or two of them had missed it I would have gone for incompetence, using Occam's Razor as a justification. However, in this case the fact that they all failed to report it to their paying customers smacks of collusion. If I were one of their customers, I would be demanding that the legal eagles look into the A-V companies' role in this affair.

Finally the whole affair raises to a high profile one of the most fundamental questions. Who owns your computer? You may think that having paid for it, you own it, but there are other contenders for this honour. In fact there are three completely different groups that lay claim to your computer – Microsoft, the media conglomerates, and last but not least, you. Waiting in the wings for an opportunity to put in its own claim is your government.

The problem is, you see, that general purpose computing machines are just that – general purpose – and this means they can be configured to do just about anything. This is great from your point of view as the owner, but a total nightmare for all the others. Microsoft relies on being able to control the operating system to lock you into its products. The media companies rely on control of the creative assets to make lots of money, and the government is generally uneasy about what you might do with all that computing power – look at the struggle over encryption, for instance.

This three-way (maybe four-way in the not too distant future) struggle isn't going to be resolved in a hurry. And really it's only the latest episode in an saga going back to at least mediaeval times when control over people's lives was contested for by monarchs, trade guilds and the mob.

As the Christian Bible so aptly puts it: *...and there is no new thing under the sun.*[1]

...Alan John Lenton, you are charged with the possession of an unregistered computer running the illegal Linux operating system. You, and your computer, will be taken from this place to the Portmeirion High Security section of our Tremadog Bay interrogation facility. Once you have confessed, you will be sent to the Epsom Salt Mines and subjected to an intellectual property rehabilitation program.

"Alan, Alan, wake up, it's time to get up!"

Alan Lenton

1 The Book of Ecclesiastes, Chapter 1, verse 9

Professionalism in Programming #36

Together We Stand (Part Three)

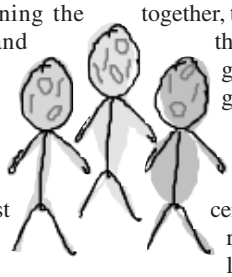
by Pete Goodliffe <pete@cthree.org>

Coming together is a beginning, staying together is progress, and working together is success.

Henry Ford

We're a long way into our voyage of teamwork discovery. We've established a practical backdrop for good teamwork, and looked at the characteristics of failing projects. Of course, not every team is doomed. So now let's see how to make some sense of this mess and how to *Do Things Right*.

In this final instalment, we'll come into dock by defining the principles of good collaborative software development, and investigating the life cycle of a healthy development team.



Teamwork Principles

First, let's look at techniques that will improve our software development teams. Although the tools and technology we discussed in part one do help to improve productivity, the largest gains are related to the human aspects of relationships between people, and their work.

First, here are Pete's Practical Principles for Programming Performance: a few key principles that, once absorbed into your group's DNA, will change the way you write software. But remember: for these to be effective you must make a purposeful change towards them; don't just agree they're good ideas and carry on coding as you always have.

Code Ownership

Many programmers are territorial about their work. This is natural: programming is a very personal, creative act. We're proud when we craft an elegant module and don't want anyone to trample all over it, destroying our masterpiece. That would be sacrilege.

But effective teamwork demands that we shed egos before entering the software factory. Don't complain that "Fred fiddled with my code". It's a team effort; the code is not 'owned' by you, it's 'owned' by the team. Without this attitude, each programmer builds their own empire, not a successful system.

With this culture in place, the team immunises itself from the danger of little Programmer Kings, each ruling their own islands of code. If no one has ever been allowed to see their code, what happens when they leave the project? Losing a local expert will severely disadvantage the team.

It's not wrong to feel a sense of parental responsibility for the code you produce, to be protective of it, and to want to nurture it. But this must be married to a healthy team focus. Instead of ownership, consider code stewardship. Stewards don't 'own' their charge, they are appointed to maintain on behalf of the owner. A steward has primary responsibility for a piece of code's upkeep, weeding it and tending the borders. Usually the steward makes all changes, although trusted team members can also make changes, which would ultimately be verified by the steward. This is a constructive approach to your code and one that will serve the team well.

Respect Other People's Code

Even in an enlightened development culture without code ownership, you must still respect other people's code. Don't tinker with it at whim. This holds especially true if they're working on it right now. You can't change something under another programmer's feet; it will cause them untold confusion.

Respect for other's code means that you should honour the presentation style and design choices currently in place. Don't make gratuitously inappropriate modifications. Honour the method of error handling. Comment your changes appropriately.

Avoid making quick hacks that you'd be embarrassed to see in a code review. They slip in when you need to get your code working quickly and one small tweak elsewhere makes your stuff compile. If you forget to tidy the tweak then you've just degraded someone else's code. Even temporary modifications must show respect.

Before you modify code, ask the steward's permission and let them know what you're going to do. If you don't, you must definitely tell them afterwards so they can review the change, and won't be surprised by it.

Code Guidelines

For collaborative development to produce reasonable code your team must have a set of code guidelines. These are dictates on the standard of code a programmer can write, ensuring that everything in the system reaches a certain minimum quality.

It's not important to stir arguments over code layout (although it is better if all code follows one style). However there must be consensus on the standard and mechanism for code documentation, for language use and common idioms, for the act of interface creation, and architectural design.

Teams that get by without such guidelines still do have them: just as unwritten conventions. The problem with such implicit knowledge is that a new team member's code won't match the existing codebase until they are integrated into the code culture.

Define Success

To feel like they're achieving something and that they're working well together, the team need a clear set of targets and goals. This must be more than milestones on a project plan, although milestones *can* be a good motivator – define lots of small milestones as short-term goals, and celebrate when you hit them.

You must define the criteria for success, so the team knows what it looks like and how to reach it. What does 'success' mean for your current project? Is it work delivered on time, to a certain 'quality'[1], with a satisfied customer, bringing in a particular revenue, or with a certain bug count? Prioritise these factors, and let the programmers know the main motivator behind this piece of development work. It will change what they do and how they do it.

Define Responsibility

All effective teams have a well-defined structure with clear responsibilities. This doesn't mean that your team has to be hopelessly hierarchical with a strict pecking order and multiple levels of management. The team structure must just be clear and recognisable.

- Who has the final say on important decisions: who maintains the budget, who makes hire/fire decisions, who prioritises tasks, who approves designs, signs off code releases, manages the schedules, etc.? These are not all necessarily roles *within* the team, but they are all roles that the team must know about.
- Where does the buck stop, and whose head will roll if the project is an unmitigated disaster?
- What is each member's *responsibility* and *accountability*? What have they been assigned individual authority for, what is expected of them, and to whom are they accountable?

Avoid Burnout

No team should have impossible goals. Sanity check the project you're embarking on – there's nothing less motivating than knowing you're going to fail before you begin.

Watch how the work is split between programmers. Avoid giving all the difficult work, or all the high risk work, to a few individuals. This is a common fault, especially when a team cultivates Programmer Kings. If they burn themselves out working many extra hours, or worrying about the implications of a mistake, they'll jeopardise the project and demoralise the team.

Congratulate the team when they do well and work hard. Do it publicly. Keep feeding them praise and encouragement. It's surprising how refreshing some support and enthusiasm is.

Mix people's jobs up; don't force someone to repeatedly do the same kind of task until they get bored and give up. Give everyone a chance to learn and to grow new skills. "A change is as good as a rest": even if there's no chance to slacken the development pace, a little variety can prevent programmer burnout.

The Team Life Cycle

It's important to see our software teams in the light of their entire life. Teams don't spring out of holes in the ground, and they don't last forever. There are four distinct stages of a team's life. At each stage the focus of activities is different. We'll look at these in turn below:

1. creation,
2. growth,
3. work, and
4. closure.

1 And how will you measure this?

Sometimes you might iterate through these a few times in different orders, but every team will go through each stage. Subteams within the main development project team will undergo a similar process; this is a recursive model.

Team Creation

There is a new project looming. It needs a development team. *On your marks. Get set. Go...* A leader is appointed by the powers that be, and it is his responsibility to pull the team together. Members may be drawn from other teams, or hired specifically for this project. Wherever people come from, they have to fit together as an effective team – the success of the project (and the leader's job) depends on it!

So it all starts here. Formation establishes the core team members. At this early stage the team has not begun working in earnest yet, nor has it jelled together properly. There are a number of important considerations as the team is forged:

- You must establish where the team sits in the organisational food chain. Which other teams will it interface with? Set up communications channels with them, so it's clear how work will flow between departments and who the contacts are.

  Think about this carefully, and try to minimise communication across team boundaries to make work as simple as possible. At this stage you can design your team to have the most chance of success by eliminating unnecessary bureaucratic overhead.
- To be effective the team requires competent, talented members who have potential to become a single high-performance unit. They must cover all critical areas of experience and expertise *before* it's needed, otherwise development will stall whilst another person is sought. Plan to grow the team as required, and work out when you'll need to start looking for more people.
- Chose and communicate an appropriate team-work model, otherwise the team will adopt an ad-hoc structure and chaotic working practices. Arrange the team structure to eliminate management overhead and internal communication paths, keeping things as nimble as possible.

The initial aim when forming a team is to create more than a mere *group*. We don't need another collection of people or a little social club; we need a cohesive, a working unit who are motivated and aiming towards a single common goal.

Don't bring a team together until you *really* know what it exists to do. If people are drafted in and then start by sitting on their hands, waiting for their instructions, then the team's long term ethos will be to hold back; there will forever be untapped potential. If the team can't begin working from the outset, don't bring it together yet.

Team Growth

After creation, once the team is populated with core staff, the project will begin to gain momentum. The team must grow to accommodate the increased workload. There are several facets to this: the team must grow in numbers, but also in experience and in vision. It must grow *outwards* and grow *inwards*.

Inward Team Growth

As they work together, the members get to know each other on a personal and professional level. The team settles into a work pattern and a coding culture is established. At first, this must be subtly guided so the culture is healthy and will serve the team structure and goals.

This stage aligns personal and team objectives, and determines the individual roles and relationships. The team's 'feel' at this point sets the tone for the whole project, so watch out for scepticism or bad will.

If it hasn't already been provided, the team infrastructure is laid down as the work builds up. Tools like source control and groupware are deployed. The project specifications are written, objectives are solidified, and the scope of the work is determined.

Outward Team Growth

Outward growth sees the accretion of more members – this is the visible kind of team growth. At it's zenith, the team contains each of the following roles. These are not necessarily individual job titles; it depends on the size of the team. In a small team, individual members take on more than one role, either full or part time. Large projects may have whole departments per role.

- *Analyst* – also called a *problem domain specialist*, this is the liaison between the programming team and the customer. He studies and understands the Real World problem well enough to write a specification that the developers can implement.
- *Architect* – a high level design authority who devises a system structure based on the analyst's requirements.
- *Database administrator* – designs and deploys the database infrastructure for the project.
- *Designer* – works below the architect to design components of the system. This is often a facet of the programmer's job.
- *Programmer* – naturally the most important person on the entire team!
- *Project manager* – takes overall responsibility for the project, making crucial decisions. The manager balances contending project forces (e.g. the budget, deadlines, requirements, feature set, and software quality).
- *Project administrator* – supporting the manager, deals with the day-to-day running of the project team.
- *Software quality assurance engineer* – produces QA plans and ensures the code produced is of an appropriate standard
- *User educator* – writes product manuals, ensures marketing is accurate, draws up training schedules, etc.
- *Product delivery specialist* – or *release engineer*, plans how to package, manufacture, distribute, and install the final product.
- *Operations/support engineer* – supports the product 'in the field', once it's in the hands of end users.

A successful project must make sure that all these activities are covered. As the need for each role is felt, people must be brought in before the need is acute. Appointing members needs management insight, both of a candidate's personality type, their technical skill, and the job requirements. Now the team is established, new people must match the working practices and compliment existing team members.

The difference in quality of developers has been shown to be very large, so try to get good people and pay them accordingly.

Work

This is the point of performance, when the team is functioning fully with everyone in place. The cogs turn and the software construction process grinds relentlessly onwards.

The majority of a team's life is spent in this phase, working out the project's objectives. To do this, the single large task is decomposed in to a series of smaller tasks. Team members are assigned their own work packages, and kept synchronised (perhaps by a project meeting, or by close communication). Their work is integrated as it's completed. Slowly the software takes shape.

Although working to a predetermined development process, the team must adapt to changes as they arise, handling unforeseen problems, changes in the team, or the dreaded Shifting Requirements Syndrome. As work progresses each member must identify and manage outstanding issues and risks.

The team must get into a development groove – finding the appropriate pace of work, and meeting targets at each step of the way. However, you must prevent the grove from turning into a rut. Don't be frightened to shake up working practices – if required – to ensure that the team doesn't get complacent or lazy, or to counter ineffective team members who might jeopardise progress.

Closure

Eventually, even the most delayed project will come to an end. That end might be successful software that makes the customer happy; it might be a doomed product and prematurely abandoned development. Either way, the project concludes and the team is removed from it.

From the very beginning of development, a clear end point must be in sight. No team can continue forever, or plan to work indefinitely. A successful project expends huge amounts of energy; you can't continue like this, or it will lead to burnout or boredom. The lure of completion actually motivates people, and many programmers won't invest much effort until confronted with a hard deadline.

For this reason every team must plan to disband, dissolve, or transition to a different kind of team (perhaps a maintenance or support team) upon project completion. This plan must cover both normal and abnormal completion conditions.

Team disbanding doesn't happen suddenly. Projects don't halt without warning; they slowly ramp down. Usually we transition people off a project gradually as they become surplus to requirements. No team needs people kicking around doing nothing, absorbing resources. As each person leaves the team, ensure that all their important knowledge and work products are captured. It's easy for information to leak between the cracks of a splitting team.

What happens once a team gets to the end of a project? The next step could be to:

- move the team into support mode, maintaining the product,

[concluded at foot of next page]

On Killer Apps

Ian Bruntlett <ianbruntlett@hotmail.com>

I have been thinking and reading avidly about three topics – thinking, creativity and schizophrenia. All of a sudden they unified into a common ground and I noticed that they explained the presence and absence of Killer Apps (ground breaking IT systems that changed things forever) in I.T. projects.

Lateral Thinking, Creativity.

In the 80's a friend of mine, observing me working noted that I was employing "lateral thinking". Later on, when I wasn't so broke, I discovered Edward de Bono's books and that got me to think about thinking which later on diversified into thinking about creativity. The magazine New Scientist (NS) has run a number of special issues this year, digging deep into how we think and how we create. A few things caught my eye. First NS stated that the creative process applies as much to science as it does to the arts. A bit obvious, maybe, but it caught my eye.

Schizophrenia

Secondly, NS noted that creativity is the flip side of the coin marked schizophrenia. Since about 2003 I've been scouring the internet and libraries for information about schizophrenia – what it is, why have I got it, how to cope with it and how come evolution hasn't driven it out of the gene pool? Both in the hospital and in the literature, creativity has been linked with psychiatric disorders. One of the killer problems with schizophrenia is that sometimes when you think you are making sense, in fact your arguments come across as rather confused. I wonder how this article rates for lucidity.

Creativity – Immersion, Obsession.

I've read and thought long and hard about this. If you want to be creative in a field of endeavour, you need to build up a breadth of experience (immerse yourself) and dig deep into the problem domain (obsession) – for every success there can be 99 failures – but you don't notice the failures, you just keep on working. And you also need to take inspiration from a multitude of sources, not just those that are similar to what you already know.

Programming and Killer Apps

I've worked on a couple of killer apps in the past. The fundamentals of creativity (immersion, obsession), applied to programming, are needed to get the project off the ground. However, the economic opportunity to employ the creative programming skills is also needed. To summarise, you need:

1 A fertile technology base.
2. The economic opportunity to shine.

With a technology base and no economic opportunity you end up on a plateau, peering over the edge, wondering what to do with all the technologies and skills you have acquired – this isn't so bad because you can beaver away, waiting for the economics to fall into place.

With a poor technological base and good economic opportunities, you're in for an awful time and the sooner you can escape these circumstances, the better. Looking back, when I've experienced that, I should have briefed my manager about the technology gap and work with them to close that gap. Sadly, I was too busy coding my way out of a problem instead of thinking my way out.

With (1) and (2) in the bag, you have the opportunity / sweet-spot for a killer app. You may be too busy to notice it, though.

At one point in my career, I would go for a drink with friends after work, nestling a copy of Charles Petzold's *Programming Windows 95* under my arm and spend evenings working on infrastructure to port an established product (public libraries search engine) to Win32 – if you're working that hard, you should also set aside time to take a break from work.

If you are working on an established killer app, you need to take time out to learn new technologies and to revitalise existing products and develop new products. Good luck.

Ian Bruntlett

[continued from previous page]

- start some new development work (perhaps a new version of the same software),
- instigate a post mortem if the project was a failure, or
- split the team up to work on separate projects (or release them if their contract expires).

Whether a team is recycled or disbanded is a difficult choice, and one that's often made badly:

- Just because a team was successful on one project doesn't mean that they will be on the next. A new project may require a different mix of skills, or a different development approach.
- However, it's wise to keep a good team together. Well integrated teams with competent members and an effective work culture are rare. Don't throw them to the wind needlessly.

When there's a choice, it should be made based on the characteristics of the next project. Sometimes this choice is made for you, though: in small development organisations the project team *is* the whole development team. It's simply not possible to mix and match programmers and you are forced to use same people on the next project.

People Power!

Finally, here are a few simple guidelines for leading a team of software developers. Without programmers you don't get programs, so we need techniques that release the potential in people and help them to work together. Even if you're not in a leadership position right now, you can use these as a simple yardstick to judge how your team is run and how people are treated. They distil a lot of the wisdom we've already seen into practical bite-sized chunks.

- *Use fewer, and better people*. Larger teams require more communication, and are harder to share vision with.
- *Fit tasks to capability, and also to motivation*. Avoid the *Peter Principle*: excellent programmers get promoted to managerial positions to which they are not suited, and not interested.
- *Invest in people*. You'll get more out of them if you build something into them. Technology moves fast; don't leave their skills out of date. Otherwise they'll move somewhere where they *will* gain better experience.
- *Don't cultivate experts*. It's dangerous when one programmer becomes the only expert in a certain area. They are a single point of failure. Some people actively try to become a Programmer King, others are forced into it, not being allowed to work on anything else. When your expert needs a new challenge, he'll leave. How will you maintain the software now?
- *Select complementary people*. The team can't all be world-class experts. They equally can't all be inexperienced programmers. You need a healthy skills mix. You also need a healthy interpersonal mix, with personalities that gel and work together well.
- *Remove failures*. Someone who doesn't fit should be removed. It's not easy to do, but a rotten part can quickly spoil the whole – the consequences of procrastination can be dire. Don't wait to see how things will pan out, or just hope they'll improve. Deal with the problem.

Conclusion

As programmers, we care about writing good code, so does all this matter? Yes: the health and structure of our software teams has a direct affect on the health and structure of our code. They are inextricably linked. Software is written by humans. Just as the software components have to fit together, communicate well, and form a cohesive structure, so must the programmers building it.

Good teamwork comes from more than a well-defined process or a fixed structure. Good teamwork stems from good individuals. "The whole is greater than the sum of its parts", or so the saying goes. This is, of course, only true if the all parts are working well. If any single part is failing then the whole will be compromised. Our individual attitudes affect the quality of our teams, and therefore the code produced. We must address these attitudes to create good code. Understanding your natural attitudes and responses will help to improve your programming skills.

A 'professional' programmer *has* to be able to work in a team. Alongside technical skills, they must be able to create a piece that will fit into the larger jigsaw. This means being able to communicate and work with others. It means understanding your role, and carrying it out appropriately, working to the best of your ability. It means cooperating with other team members; being team-focused, not self-focused.

Pete Goodliffe

Look out for Pete's forthcoming book, released in 2006!
Watch this space for details.

Interview with Charles Moir, Xara

Paul F. Johnson <cvu@accu.org>

Xara is a well known company in the field of vector graphics for its ground breaking art package bearing the same name as the company. Its products have been marketed around the world (for a while, by Corel) and are a common feature on many desktops having captured the end user with (relatively) inexpensive licences, fantastic backup and a background of innovation and reliability.

Xara is not a new company; for many years it was known as Computer Concepts where it made its fortune in the Acorn market, starting with the BBC B and moving onto the RISC OS 32 bit platforms. It also managed to stump Acorn itself by releasing its desktop publisher, Impression, at the same time as Acorn.

Recently, Xara decided to release the source code for its Xara art package into the public domain (under the GPL licence) to enable Linux and Mac users to take advantage of this formerly Windows only application.

In this interview with Charles (held over email), we discussed the development of this and why this bold decision has been made.

About the Company

Xara originally started its life as Computer Concepts, a company writing software for the Acorn market (back in the days of the BBC B, later progressing to the 32 bit Acorn machines). How big a decision was it to stop producing software for the RISC OS platform and move over to the PC world and what were the key factors governing this decision?

We had a lot of flack from die-hard Acorn enthusiasts who saw us as selling out, but it was plain obvious from a commercial point of view that Acorn were not going to make it, so we had to move platform. Plus the fact that the main reasons for us developing only on the Acorn platform in the first place were being eroded quite rapidly (It was the first 32-bit RISC platform, so from a programmers point of view it had a nice linear address space.) At the same time the DOS / Windows / Intel and even the Mac world were dealing with the horrors of segmented address space and 16-bit architectures. Even the Mac that was using a nice processor (68000) was hindered by a very primitive OS – you couldn't, for example, allocate more than 64K of memory. People forget just how nasty it all was back then. On top of that the Acorn platform had the most advanced Windows GUI of the time, and the best graphics, the best font system, I could go on. If I'd told my guys that we're stopping 32-bit programming and going to work on the Intel processor, 16-bit segmented, DOS-based world, they probably would have walked out. The fact was that we were (and have always been) successful at doing what we find interesting, not what might generate the most income. Had we been entirely commercially orientated we'd have been working on IBM PC when it came out. But really, it was just so bad from just about every technical point of view you could think of, that I had no interest in it at all.

By 1990 it was pretty clear Acorn were going to lose it, and a few years later it was inevitable, so we had to switch. At the same time Windows machines were getting a lot better. Graphics capabilities were coming on leaps and bounds. Windows NT was coming (at last a real operating system that has protected memory processes, multi-threading, 32-bit linear address space. No more segmented world.)

So it was a huge and complete change. One day were all programming on Acorn RISC platform, and then almost overnight it was mass migrate to Windows programming, albeit that we never had to mess with the segmented DOS world.

Oh, I should just say that I've always been a fan of Apple. But in the early nineties when we had to make the switch from Acorn, Apple looked as if they'd lost the plot completely. Jobs was no longer there. Apple was producing terrible beige boxes. They had no clear strategy. CEOs came and went. Ask anyone at the time where Apple would be in five or ten years and the universal answer was 'dead'. So, much as we would have liked to switch to Mac, it seemed as if that would be suicidal. Of course that was not counting that Jobs would come back, and sort them out, as he has done. So with hindsight we, I, made the wrong decision. We should have moved to the Mac (Linux didn't exist back then).

Did the move to the world of Windows change the mode of development and the primary languages used?

Although many modern programmers can't understand this, until the mid-nineties all our programming was 100% assembly language. We created very large, complex, mainstream applications such as desk-top-publishing applications, graphics applications (Artworks the predecessor of the current Xara Xtreme), programming languages, databases, utilities – just about every type of software you can think of – all 100% assembly language.

Part of the reason we could do this, it has to be said, was that programming in ARM assembly language was a lot, lot easier than in Intel assembly language. And I should say that, even though it was all assembler that didn't stop us programming in an a modular, even object orientated fashion. The core techniques of these approaches, such data hiding and encapsulation, can all be applied just as well to assembly language as C++.

But we knew we had to move to a C or C++ based world. And it was a complaint we had about the Acorn platform, that they couldn't get their C compiler act together – when they did, it was too late.

So we switched to C++. It gives enough low-level control for people who come from an assembly background, but isn't too high a level language (say like Java or C#) that is so removed from the actual hardware and processor, that you feel you have little control or knowledge about the machine code being produced (not that it matters so much nowadays).

How did the change of operating systems affect the number of employees of Xara and did it offer any great financial security than the RISC OS platform?

Well we could all see Acorn were doomed, so we had to switch. So in that sense it provided security. We decide to focus on just one application, that was a re-write of Artworks, the graphics program, which was to become Xara Studio. We put a huge amount of time and effort into this. I think we spent two to two and half years, 20+ developers, so it was a huge project.

About the Software

CC was known for a number of products, most notably the Impression family and Artworks. With the change from CC to Xara, Artworks was continued and developed in the Xara art package. Xara and Corel linked up to produce Corel Xara. How did this come about?

We targeted Corel. They were the dominant player in the graphics market for Windows. Something like 90% market share. CorelDRAW was a good product, great value for money, so they deserved to be where they were. But we knew we could produce a better product (and had already done so in Artworks on the Acorn platform). So Corel were the enemy that we had to beat. We all had a simple goal – create a better product than CorelDRAW (and Adobe Illustrator – which was very easy as Illustrator was very backward indeed in those days.)

Xara has always tried to produced the best technical products. The highest performance, the easiest to use – that's why we were so successful in the Acorn market and you can see this today in our products still. But we knew even in 1995 that we were not the organisation to market the product – we didn't have the world-wide clout. So we imagined it was probable that we'd sell it to Microsoft, or IBM or some other large world-wide distributor.

As we were getting near to completing Xara Studio we were getting previews of the product in the press, and they were very complimentary and started to say things like "at last here's a product that can topple Corel". Well Corel noticed. So they sent a bunch of execs on a plane and were told not to come home until they had a deal with us.

So to cut a long story short – we sold to Corel, the company we'd just spent a few years competing against. BUT we knew this was dangerous, so we were careful in the contract and did not sell the technology. We came to a pure marketing agreement whereby they'd sell the product under the Corel brand, and pay us to continue to develop it. We did not give them access to our source code. They wanted to buy the company – we didn't want to sell. We made sure it was financially a good deal that we couldn't lose by, even if Corel failed at marketing it as they said they would.

So for the next five years it was a Corel product.

How important was the link up with Corel and did it cause any problems when the partnership was dissolved?

We had to find someone to sell the product around the world. We'd spent nearly all our money developing the product and did not have the resources to market it on the scale the product deserved. We were actually talking to Microsoft and others, but Corel were the most aggressive and so did the deal with us first.

By the end of the nineties Corel were going through some tough times, and so the fact they were still paying us was hurting them. On top of that it was obvious that in fact their strategy was to control a competitor and threat to

CorelDRAW and so were not marketing our product as they should have been (our deal anticipated this possibility), and that was very frustrating of course. We both wanted out of the deal, so it was pretty easy really. We'd get all the marketing rights of the product back, and they'd stop paying us.

Was there any plans to continue Impression for the PC market?
Ha, yes. Impression was our very popular DTP product for the Acorn platform. It was very Quark-like in many ways (better in quite a few). In fact Xara Studio (known as Camelot internally) was designed to be an Impression and Artworks combined , a DTP and drawing program in one. Even today I see no reason why you should have to use a separate DTP program from your graphics program. And if you look at modern graphics products such as Corel, Freehand and others, they have a lot of DTP-like capabilities. And high-end DTP products have a load of graphics capabilities, even being able to create lines, fills, basic drawing and gradient fills, and of course provide a load of photo handling capabilities.
DTP and illustration packages share a lot in common – they are both really publishing tools. If you look at the page describing the basic architecture of Camelot here `http://www.xaraxtreme.org/developers/basics.html` *you'll see lots of DTP-esque features and terminology, such as spreads, chapters and more. The problem is that we've never actually managed to get around to implement most of these features, so right now it' s more a pure vector illustration tool, and rather less a DTP product. In fact the text handling capabilities of Camelot are rather poor, which is sort of ironic given the very powerful text handling features of Impression. In my very early days I developed (I think it might have been the very first) check-as-you-type spell checker for Wordwise, a word processor I wrote (this would have been 1985 type of timescale), and yet here we are 20 years later and we still don't have that feature in Xara Xtreme – and I really need it.*

Xara Xtreme is the new product. Tell us what sets it aside from other vector graphics applications.
The key characteristics that have defined Xara software from the beginning. Performance, simplicity, ease of use. We have the fastest, most powerful vector graphics rendering engine in the world today. It offers a very wide range of graphic capabilities, and yet looks a very simple product – relatively few dialogs, menus etc. It has a clean, orthogonal feature set and architecture. So, for example, all objects on the page are treated the same. So you can, say, colour a photo the same way you colour any object. You can feather the edge (give a soft blended edge) to some text, a photo or any vector graphic, using the same tool.
We've also pioneered some user-interface concepts that you'll be seeing a lot more of, not just from us. So we've always tried to go for live preview. That is where things happen in real-time as you drag or select things, and usually 'on-canvas', in the main document, and not in separate dialogs. So for example if you adjust the transparency, colour or feathering of objects on the page – these change on the page as you adjust the values. Live interactive preview.
We were doing this 10 years ago – check out the new Office 12 from Microsoft. They have finally woken up to the concept, and use it to huge effect in Office 12, and it's great. We should, and will, be doing more of this. Once you've used live-preview tools, you'll never go back, and everything else seems positively crude and clunky.

Moving to Open Source

Xara Xtreme is being dual licenced; one chargeable (Windows) and a GPL version for Linux and MacOS. What has brought this change about?
Two parts to this; Why the change to open-source?
The answer to that is because we face greater competitive threats now than at any time in our history. There is a consolidation amongst the big players (Adobe, the current number one player has acquired the number 2 player Macromedia), and a battle is looming between Adobe and Microsoft who have decided, for the first time in 25 years, to get into this space (vector / photo graphics software).
We do not want to get squashed or side-lined out of existence by the giants. And for those companies it's not about the technical merits of a product, but more about marketing clout. They both throw millions, tens, hundreds of millions of dollars promoting their products. We can't compete against that sort of money. We can more than compete on a technical front.
At the same time there is a very large gap in the Linux and open-source world for good graphics applications. In fact there is less choice and competition in the graphics market for Linux now than there was 15 years ago in the Acorn market (which was a fraction of the size). That's weird, and an opportunity. Add to the fact that within the last year or two the Linux desktop has evolved enormously, so that now it's a very good, high quality, effective desktop platform.
So why not just port the product, as a commercial product, to Linux?
People have tried and failed. If you want true mass-market adoption in the Linux market, it's simple, it has to be open-source and preferably using the GPL license.So that got us thinking about whether we could somehow do this and still have a business.
At the same time being open-source does have quite a few possible advantages. One, for the customers and users, it means the product can never be eliminated (as is happening to Macromedia Freehand – a fine product with thousands of users who are just about to lose their tool). Secondly it might give us larger technical resource to better compete against the giants, Microsoft and Adobe. It hopefully is a way in which we can create a good Linux and Mac versions, which otherwise we'd not be able to afford or justify ourselves. So the ideal scenario is a win-win for everyone. The Linux and open-source community get a great, very slick, stable, commercial-quality graphics product, that's GPL, so they can do what they like with it. We get a Linux and Mac port of the product. We hope that the combined resources of Xara and open-source developers will progress the product to create a really competitive alternative to Microsoft's new graphics products and Adobe's. We hope that we can continue to sell commercial version into the Windows market (and perhaps, if there's enough demand, into the Linux and Mac market as well).
Anyway we came to a decision that the potential upside was worth the risk. The risk is that we lose control of the product and we lose revenue by 'giving the product away' as it were. These are very real and substantial risks.
So, we hope to survive as a business, by going for a dual license, so we can continue to sell commercial versions, probably mostly into the existing Windows market as we have, and a GPL version for Linux and open-source platforms.

How do you intend to protect the value of the Xara Xtreme software (for example, as the source is GPL and based on wxWidgets, there would be nothing to stop someone recompiling for Windows)?
That is the key risk. There's nothing to stop anyone using the GPL code we release, and creating a Windows version and competing against us. And if, as a result, our Windows income decreases, and we can't make it up any other way, then we've got a problem. The whole experiment and the gamble could well be a failure. So although we can't stop it, we hope that most developers will be more interested in creating Linux, BSD, Mac, Solaris, or RISC OS even, ports, than they would creating a Windows version. We already have a Windows version and it costs only $79 (less than £50) and so it's, frankly, just a waste of effort creating a Windows port when that same developer effort could go into adding a new feature that benefits everyone – and makes it more competitive against the Microsoft equivalent.
Our commercial version does have benefits. The money gets you things like a CD, manual, and a bunch of licensed third party plug-ins and closed-source features (that we can't open source anyway), and customer support and other things.

As with the likes of the Fedora Core / Red Hat Enterprise Linux relationships, will developments from the Open Source version be ploughed back into commercial version (and where possible, vice versa) and if that is happening, what protection is there on both sides against patent and copyright problems?
That's the plan. Right now we've got full-time programmers working on porting this product to try and make it cross-platform and open-source. If, as a business, we continue, perhaps even thrive, we'll continue to invest more into the development of the product. Right now all bug fixes and features added to the Windows build are immediately reflected in the open-source code base. The plan, and it's probably the only practical goal in the end, that we have one code-base that covers all platforms, so that when anyone adds a feature it appears, with as little effort as possible, on all platforms.
As to the patent issue, this is a bugbear of mine. We've innovated for 25 years or more, with dozens, probably hundreds, of ideas, techniques that we pioneered or invented. Imagine if I'd patented spell-check-as-you-type, – well it's a ridiculous idea (was at the time, nowadays people seem to think they should have a monopoly on ideas, almost all of them incredibly obvious ideas to experienced software developers – hasn't stopped them being patented, at least in the US). We are gracious enough to understand that everything we do – indeed ALL software is built on what went before. All software is an evolution – taking the existing ideas and improving on them,

by inventing new ways to do things to improve on existing techniques. I believe it's the principle job of software engineers to solve problems, by inventing solutions, and the concept that they should keep those ideas to themselves is absurd and, well, offensive.
Software patents are a government granted monopoly. Monopolies are bad – it's that simple. They are bad for users and customers, bad for innovation, bad for competition. Patents stifle competition – and competition is what's driven the huge growth and success of the entire IT industry for the last 25 years. Some people (mostly lawyers with vested interests) want to stop this. For almost the entire history, a period of explosive growth, of the software industry, everyone shared their ideas. There were no software patents and, as a result, we got the IT revolution. Now people are trying to stop this with software Patents.
(That is not the same as copyright – copyright is very different. Everyone should have the right to keep their actual writing or expression of their ideas to themselves. That's what the copyright law gives everyone. That's completely different to the patents, which try to monopolise the ideas itself). So of course we fully respect, and our entire business is build around, the copyright law.)

Why are you using the GPL and not one of the other Open Source licences?
Two main reasons. It's the most accepted open-source license, and secondly what's sometimes referred to as the copyleft aspect of this license. The GPL licence prevents our commercial competitors using our source code against us. They can't use any of our code in their products, without making all their source code of this product open and public as well. That's not going to happen at Adobe or Microsoft. They are public companies with shareholders and simply could not afford to do this. So this stops commercial companies using our code and technology to compete against us.

Programming

What form of version control software do you use and why?
Subversion. We looked around and the general consensus was that this was better than say CVS.

Which language is Xara Xtreme programmed in and what was the main driver behind using wxWidgets instead of (say) Qt?
We needed a cross-platform UI framework that was as native as we could get on each platform, that was open source, that has as few commercial restrictions as possible (bearing in mind that we're creating commercial versions of Xara Xtreme). That pretty much narrows it down to wxWidgets. GTK is certainly not there yet. Qt is a serious alternative, but sort of emulates the native look on some platforms, which we're not so keen on, and it also has commercial developer costs that simply do not exist for wxWidgets. Now wx is not perfect by any means – it has more than a few rough edges and it looks as if we're going to have to be doing quite a lot of custom widget work, which we'll put back into wxWidgets if it's any use to others, in order to get the really slick, best possible native look and feel that we need.
It's almost all C++ and we're trying to make it as compiler portable C++. There were / are some assembler chunks of code in the Windows version. Obviously this is not portable and not going to help our plans to make this cross-platform, and so we've put a lot of effort into creating C versions of this code, and this is what's being made open-source. There is some performance hit, about 20% in some cases, but it shows modern compilers are very good. In fact for some specific cases on the Centrino chip we've found the compiled C code runs faster than our hand-tuned assembler code. That was a surprise.

Do you use any form of agile techniques in your company?
Not really. I'm hoping that some of this will come as a natural consequence of going open-source. Having said that, we have always worked in a iterative manner. We learned a long time ago that you can never specify everything up-front, especially when it comes to user interface matters. Often it's only when you've got a feature working or half working that you realise better ways of doing it, so you go around the loop again.
Oh and we've always followed some agile-like ideas. We don't have large separate testing teams or departments. We test as we go, 'we eat our own dog food' as the saying goes and use the current build to do real work. Developers are primarily responsible for their own testing. We fix bugs before we move on to new features, that type of thing. We implement features largely because we'd like to see them, because we want to use them, rather than being driven totally by customer demand. Oh and I'm a great believer in the KISS principle, for just about everything from coding to design to user-interface work. Developers often over-engineer solutions, when there are nearly always simpler ways of doing things when you think about it. Simple is always better.

What does Xara look for in a potential employee? Does "worldly experience" count over academic qualifications?
Worldly experience any day. Over the 20 years we've employed hundreds of developers and there is often little resemblance between qualifications and skill. We've had the most qualified PhDs, with fantastic theoretical knowledge, but who when it came down to the practicalities of getting things done barely know how to work a computer, and completely lacked common sense to solve the most mundane day-to-day programming problems. You know you get some academics, in all disciplines, who get engrossed in the pure academic side of things and start to leave the real-world behind because they have so little real-world experience.
On the other hand I know some very, very talented developers who hardly have any qualifications. Some people just have a natural talent.Having said that, the situation has changed over the years. 20 years ago it was a lot more common not to have university education. So nowadays we do look for university graduates, and preferably from a good university. But if anyone has a track record and can demonstrate the talent, then lack of qualifications does not stand in the way at all.

To you, which is more important – clarity of code or clarity of thought?
They go together do not they not?
I certainly prefer simple, more verbose code than 'clever' code that might be more compact code, but less immediately understandable. The ability to stand-back and take in the whole picture and the overall goals is a rather rare talent. I find most developers (probably true ofmost engineering) over-engineer solutions, when there is nearly always a simpler and thus better approach to be taken.

How does XaraXtreme fit in within the open source market? There are other vector graphics packages (such as Inkscape and Sodopi) with a large following already.
Well it's not yet in the open-source market because it's not yet at state that it's useful even. But we'll see when we've got a finished product (by which I mean the same standard as the commercial Windows version). But that is one difference we expect to see; products such as Inkscape and many open sourced product are in such a continual state of development that they are not always the most stable and just never get to the same stable 'finished' state that commercial products do. The urge of the developers is always to move on to the next most interesting thing, which is fair enough, but that discipline is different to releasing very, very stable commercial products.
We hope that by combining the Xara commercial release approach and our standards we can get the best of both worlds. Our developers, because they are paid, work on what we believe is the most important for a stable commercial release, and that is often far from interesting work. Chasing bugs down that to most people are not that important – are that important to us. Whereas most developers usually get something to the point that they think 'that's good enough' we almost always do not think is good enough and, at least for our paid developers, ask them to go back and continue working on it in order to get it to a higher better standard than 'just good enough'.
Secondly we can plan and implement major architectural changes to the product that is sometimes difficult to get done in the open source world. Jobs that do not provide instant, or even near term gratification that most open source developers like to work on.
Thirdly Inkscape, for example, is dramatically better on Linux than it is on Windows. This is not surprising because almost all the developers are Linux users. They've had serious bugs on Windows platform that have been there for months, that no one has the incentive to fix. Users can work around it with difficulty, so it just doesn't get fixed. That standard of work would be unacceptable and untenable for any professional quality commercial product (users would be asking for their money back – they don't of course because it's free, but I believe most users therefore reduce their quality expectations because of this.) We hope and would like to aim that not just the Window version (that's where all our paying customers are), but also the Mac version are as slick, as tested as 'finished' as we can make it. Of course saying that is one thing, we'll have to see how we get on to see if it turns out that way.
One things that I find rather surprising, is that comparing the market with the early days of the Acorn Archimedes, there is less choice of graphics software on Linux than there was for the Acorn machine 15 years ago, and when there was perhaps only a hundred thousand users on the platform. Why is it that Linux has so little choice? But the fact that there might be two

[concluded at foot of next page]

Idiomatic Expressions in C

By Adam Petersen <adampetersen75@yahoo.se>

Patterns exist at all levels of scale. At their lowest level they are referred to as idioms, as they depend upon the implementation technology. This article will present a collection of such idioms for the C language.

The Evolution of Idioms

As a language evolves, certain efficient patterns arise. These patterns get used with such a high frequency and naturalness that they almost grow together with the language and generate *an idiomatic usage* for the practitioner of the language. The resulting idiomatic expressions often seem peculiar to a newcomer of the language. This holds equally true for both natural and computer languages; even a programmer not using the idioms has to know them in order to be efficient at reading other peoples' code.

The idioms at the lowest levels are useful in virtually every non-trivial C program. Unfortunately idioms at this level are seldom described in introductory programming books and more advanced literature already expects the reader to be familiar with the idioms. The intention of this article is to capture some of these common idiomatic expressions.

Idiom Description Form

Because of their relative simplicity, devoting an article using a full-blown pattern form to a single one of these idioms would be to complicate it. On the other hand, simply listing the idioms would overemphasize the solution aspect. Therefore, each idiom will be introduced with a code sketch that illustrates a certain problem context, where after the idiom is applied to the code in order to solve the problem. This before-after approach also identifies certain code constructs as candidates for refactoring towards the idiomatic construct.

The idioms presented here are of two, overlapping categories:

- *Idioms for robustness:* The idioms in this category arose to avoid common pitfalls in the C language. For example, the idiom `INITIALIZE COMPOUND TYPES WITH {0}` provides a tight and portable way to initialize **`struct`**s and **`array`**s.
- *Idioms for expressiveness:* Writing code that communicates its intent well goes hand in hand with robustness; making the code easier to understand simplifies maintenance and thereby contributes to the long-term robustness of the program. For example, the idiom `ASSERTION CONTEXT` makes assertions self descriptive.

SIZEOF TO VARIABLES

Problem Context

In the C language, generality is usually spelled **`void*`**. This is reflected in the standard library. For example, the functions for dynamic memory allocation (**`malloc`**, **`calloc`**, and **`realloc`**) return pointers to allocated storage as **`void*`** and do not know anything about the types we are allocating storage for. The client has to provide the required size information to the allocation functions. Here's an example with `malloc`:

```
HelloTelegram* telegram =
 malloc(sizeof(HelloTelegram));
```

Code like this is sneaky to maintain. As long as the **`telegram`** really stays as a **`HelloTelegram`** everything is fine. The sneakiness lies in the fact that the **`malloc`** usage above contains a subtle form of dependency; the size given must match the size of the type on the left side of the assignment. Consider a change in telegram type to a **`GoodByeTelegram`**. With the code above this means a change in two places, which is at least one change too much:

```
/* We must change the type on both sides of
the assignment! */
GoodByeTelegram* telegram =
 malloc(sizeof(GoodByeTelegram));
```

[continued from previous page]

good drawing programs on Linux is not a problem for us or anyone.
Other difference are that Inkscape is fundamentally tied to the SVG file format and its features are dictated by that file spec. Well SVG is not popular on other platforms and probably never will be (in my opinion). So our ambitions are far greater, both in terms of feature set and in terms of universal cross-platform support PDF is native vector formaton Mac and Adobe Illustrator is the industry standard vector format. So we have to support those as well as we possibly can. Again Linux users care a lot less about these types of things.
We are not bound or restricted to any file format and will add and extend the capabilities of our program as we see fit. We always have and have a history of leading the industry standards. We introduced such things as vector anti-aliasing, vector transparency and a load more years before others took it up – even though they are now industry standards. The same will be true for things like vector soft shadow and feathering, embedded JPEGs, on-the-fly (zero memory) image processing, graduated transparency. We're shipping features like this today that are simply beyond the state of the industry standard and any industry standard file specification.

Given that the commercial version has aspects which are not releasable (such as Pantone support), how much of XaraXtreme will be available as a whole to the Open Source arena?

Well I think I'm right in saying the only third party licensed code that we can't open source are a bitmap conversion library we use (easily replaceable with rather better open source ones), our PDF import / export filters, Pantone, and a range of example Live Effects Photoshop plug-ins that we license from various developers. Other than our current intention is to release all of Xara Xtreme source code. But that process will be a staged process.

You've now tasted Mac and Linux development and development techniques. How does this compare with how you have seen Windows software development take place?

Well it has to be said our Windows developers feel that MSVC is a better more productive development environment than they've found on Linux or the Mac for that matter. But perhaps part of that is simply what they are used to. Right now it also looks, I'm afraid to say, that the GCC compiler is not a match for the Visual C++ Microsoft compiler in terms of the performance of the resulting code. It's early days, but there are signs that for some areas MSVC produces a lot faster code.

Have you given any thought to using C# as a full cross platform alternative (compile once, run on all scenario)?

Yes, and personally I really like the looks of the C# language. At the same time be rather wary of any interpreted byte-code based language. No matter how good your JIT compiler is, it ain't the same as C or C++. Remember we have a background in assembler programming and the further removed you are from the hardware the more dangerous in many ways. I'm already very aware that 'modern' programmers who only learn high level languages really have no concept of what's going on 'under the hood'. And I certainly find that very few developers nowadays have any concept of just how fast their processors really are or what should be possible. Our DTP product on the ARM was as fast on a 10Mhz ARM as many modern equivalents are on 1000Mhz processors, and that's including a full GUI desktop and outline fonts and the lot.
Only by being an assembler programmer with an inherent understanding of memory bandwidth, cache, instruction sets etc. do you really appreciate what is possible (and also explains why what most programmers regards as 'very fast' I usually regard as very slow.) There should be a law, there probably is, that says "developers always use all the space (RAM) and speed (MHz) you're given, and then just a bit more, to do the same job".
Having said that, if I was starting a new project now (Xara Xtreme is not new so it's not an option), I'd almost certainly do it using C#. Mono is a great project and deserves complete support in the Linux community. If only a good Mac C# (and associated libraries) existed then I think this would be the best high-level, most productive language around.

Charles, thanks for your time in doing this interview.

Paul Johnson

A failure to update both places may have fatal consequences, potentially leaving the code with undefined behaviour.

Applying the Idiom

By following the idiom of SIZEOF TO VARIABLES the dependency is removed. The size follows the pointer type being assigned to and the change is limited to one place. The original example now reads:

```
HelloTelegram* telegram = malloc(sizeof
 *telegram);
```

But wait! Isn't the code above dereferencing an invalid pointer? No, and the reason that it works is that `sizeof` is an operator and not a function; `sizeof` doesn't evaluate its argument and the statement `sizeof *telegram` is computed at compile time. Better yet, if the type of `telegram` is changed, say to a `GoodByeTelegram`, the compiler automatically calculates the correct size to allocate for this new type.

As `telegram` is of pointer type, the unary operator `*` is applied to it. The idiom itself is of course not limited to pointers. To illustrate this, consider functions such as `memset` and `memcpy`. These functions achieve their genericity by using `void*` for the pointer arguments. Given only the `void*` pointer, there is of course no way for the functions to know the size of the storage to operate on. Exactly as with `malloc`, it is left to the client to provide the correct size.

```
uint32_t telegramSize = 0;
memcpy(&telegramSize, binaryDatastream,
 sizeof telegramSize);
```

With SIZEOF TO VARIABLES applied as above, the size information automatically matches the type given as first argument. For example, consider a change in representation of the `telegram` size from 32-bits to 16-bits; the `memcpy` will still be correct.

INITIALIZE COMPOUND TYPES WITH {0}

Problem Context

Virtually every coding standard bans uninitialized variables and that with very good reasons. Initializing a basic type such as `int` or `char` is straightforward, but what is the correct way of initializing a compound type like an `array` or `struct`? A dangerous but unfortunately common practice is shown below:

```
struct TemperatureNode {
   double todaysAverage;
   struct TemperatureNode* nextTemperature;
};

struct TemperatureNode node;

memset(&node, 0, sizeof node);
```

The problem is that `memset`, as its name indicates, sets the bits to the given value and it does so without any knowledge of the underlying type. In C, all bits zero do not necessarily represent floating-point zero or a `NULL` pointer constant. Initializations using `memset` as above result in undefined behaviour for such types.

The alternative of initializing the members of the struct one by one is both cumbersome and risky. Due to its subtle duplication with the declaration of the struct, this approach introduces a maintenance challenge as the initialization code has to be updated every time a member is added or removed.

Applying the Idiom

Luckily, the portable solution provided by the INITIALIZE COMPOUND TYPES WITH {0} does not only ensure correctness; it also requires less typing. The code below guarantees to initialize all members (including floating-points and pointers) of the structure to zero. The compiler guarantees to do so in a portable way by automatically initializing to the correct representation for the platform.

```
struct TemperatureNode node = {0};
```

At the expense of creating a zero-initialized structure, `memcpy` may be used to reset an array or members of a structure later. Because it works by copying whole bytes, possible padding included, `memcpy` does not suffer from the same problem as `memset` and may safely operate on the structure in our example.

```
const struct TemperatureNode zeroNode = {0};
struct TemperatureNode node = {0};

/* Perform some operations on the node. */
  ...

/*Reset the node (equal to node = zeroNode; )*/
memcpy(&node, &zeroNode, sizeof node);
```

Using `memcpy` for zeroing out a struct sure isn't the simplest possible way. After all the last line above could be rewritten as `node = zeroNode` while still preserving the same behaviour. Instead the strength of this idiom is brought out when applied to an array. It helps us avoid an explicit loop over all elements in the array as `memcpy` now does the hard and admittedly boring task of resetting the array.

```
const double zeroArray[NO_OF_TEMPERATURES]
 = {0};
double temperatures[NO_OF_TEMPERATURES] = {0};

/* Store some values in the temperatures
array. */
...
/* Reset the array. */

memcpy(temperatures, zeroArray,
 sizeof temperatures);
```

ARRAY SIZE BY DIVISION

Problem Context

The C language itself does not provide much support for handling its built-in arrays. For example, when passing a certain array to a function, the array decays into a pointer to its first element. Without any specific convention and given only the pointer, it simply isn't possible to tell the size of the array.

Most APIs leave this book-keeping task to the programmer. One example is the `poll()` function in the POSIX API. `poll()` is used as an event demultiplexer scanning handles for events. These handles, which refer to platform specific resources like sockets, are stored in an array and passed to `poll()`. The array is followed by an argument specifying the number of elements.

```
struct pollfd handles[NO_OF_HANDLES] = {0};

/* Fill the array with handles to poll, code
omitted... */

result = poll(handles, NO_OF_HANDLES, INFTIM);
```

The problem with this approach is that there is nothing tying the constant `NO_OF_HANDLES` to the possible number of elements except the name. Good naming does matter, but it only matters to human readers of the code; the compiler couldn't care less.

Applying the Idiom

By calculating ARRAY SIZE BY DIVISION, we are guaranteed that the calculated size always matches the actual number of elements. The calculation below is done at compile time by the compiler itself.

```
struct pollfd handles[NO_OF_HANDLES] = {0};
const size_t noOfHandles
 = sizeof handles / sizeof handles[0];
```

This idiom builds upon taking the size of the complete array and dividing it with the size of one of its elements (`sizeof handles[0]`).

MAGIC NUMBERS AS VARIABLES

Problem Context

Experienced programmers avoid magic numbers. Magic numbers do not communicate the intent of the code very well and may confuse anyone trying to read or modify it. Traditionally, some numbers like 0, 1, 3.14 and 42 are considered less magic than others. Consider a variable that is initialized to 0. A reader immediately expects this to be a default initialization and the variable will probably be assigned another value later. Similarly, culturally acquainted people know that 42 is the answer to the ultimate question of life, universe, and indeed everything.

The problem with all these not-so-magic numbers is that they build upon assumptions and expectations. These may be broken. One example is `struct tm`, which is used to represent the components of a calendar time in standard C. For historical reasons and in grand violation of the principle of least astonishment, assigning 0 to its `tm_year` member does not represent year 0; `tm_year` holds the years since 1900. Sigh.

Pure magic numbers are of course even worse. Consider the code fragment below:

```
startTimer(10, 0);
```

Despite a good function name, it is far from clear what's going on. What's the resolution – is 10 a value in seconds or milliseconds? Is it a value at all or does it refer to some kind of id? And what about this zero as second parameter?

Applying the Idiom

A step towards self documenting code is to express MAGIC NUMBERS AS VARIABLES. By applying this idiom to the code construct above the questions asked do not even arise; the code is now perfectly clear about its intent.

```
const size_t timeoutInSeconds = 10;
const size_t doNotRescheduleTimer = 0;

startTimer(timeoutInSeconds,
 doNotRescheduleTimer);
```

Of course the whole approach may be taken even further. By writing

```
startTimer(tenSecondsTimeout,
 doNotRescheduleTimer);
```

the code gets even more clear. Or does it really? The problem is that to the compiler the variable `tenSecondsTimeout` is really just a name. There is no guarantee that it really holds the value 10 as an evil maintenance programmer my have changed the declaration to:

```
const size_t tenSecondsTimeout = 42;
/* Original value was 10 */
```

Such things happen and now anyone debugging the program will be unpleasantly surprised about how long ten seconds really are. They may feel like, hmm, well, 42 actually.

An idiom cannot be blindly applied and MAGIC NUMBERS AS VARIABLES is no exception. My recommendation is to use it extensively but avoid coding any values or data types into the names of the variables. Values and types are just too likely to change over time and such code gets unnecessarily hard to maintain.

NAMED PARAMETERS

Problem Context

As we expressed MAGIC NUMBERS AS VARIABLES the code got easier to read. That is, as long as the names of the variables convey meaning and finding good names is hard. To illustrate this, let us return to the previous example where a timer was started and extend it to include the start of two timers. I am rather happy with the name `timeoutInSeconds` and would have a hard time finding a second name that helps me remember the purpose of the variable equally well. Instead of going down the dark path of naming by introducing communicative obstacles such as `timeout1` and `timeout2`, I try to reuse the existing variable by removing its `const` qualification and re-assign it for the second timer.

```
size_t timeoutInSeconds = 10;
const size_t doNotRescheduleTimer = 0;

notifyClosingDoor =
 startTimer(timeoutInSeconds,
 doNotRescheduleTimer);

timeoutInSeconds = 12;
closeDoor = startTimer(timeoutInSeconds,
 doNotRescheduleTimer);
```

This is a tiny example, yet it's obvious that the code doesn't read as well as before. The extra timer is part of the story, but there's more to it. By re-using the `timeoutInSeconds` variable, a reader of the code has to follow the switch of value. As the right-hand sides of the expressions starting the timers look identical, the reader has to span multiple lines in order to get the whole picture.

Applying the Idiom

By naming parameters, it is possible to bind a certain value to a name at the immediate call site. C doesn't have language support for this construct, but it is possible to partly emulate NAMED PARAMETERS.

```
size_t timeoutInSeconds = 0;
const size_t doNotRescheduleTimer = 0;

notifyClosingDoor =
 startTimer(timeoutInSeconds = 10,
 doNotRescheduleTimer);

closeDoor = startTimer(timeoutInSeconds = 12,
 doNotRescheduleTimer);
```

The `timeoutInSeconds` variable is still re-used to start the timers, but this time directly in the function call. A reader of the code is freed from having to remember which value the variable currently has, because the expression now reads perfectly from left to right.

As neat as this idiom may seem, I hesitated to include NAMED PARAMETERS in this article. The first time I saw the idiom was in a Java program. It felt rather exciting as I realized it would be possible in C as well. Not only would it make my programs self-documenting; it would also bring me friends, money, and fame. All at once. After the initial excitement had decreased, I looked for examples and possible uses of the idiom. I soon realised that the thing is, it looks like a good solution. However, most often it's just deodorant covering a code smell. Most of the examples of its applicability that I could come up with would be better solved by redesigning the code (surprisingly often by making a function more cohesive or simply renaming it). All this suggests that NAMED PARAMETERS are more of a cool trick than a true idiomatic expression.

That said, I still believe NAMED PARAMETERS have a use. There are cases where a redesign isn't possible (third-party code, published APIs, etc.). As I believe these cases are rare, my recommendation is to rethink the code first and use NAMED PARAMETERS as a last resort. Of course, comments could be used to try to achieve the same.

```
closeDoor = startTimer(/* Timeout in seconds */
12, /* Do not reschedule */ 0);
```

Because I believe that such a style breaks the flow of the code by obstructing its structure, I would recommend against it.

ASSERTION CONTEXT

Problem Context

Assertions are a powerful programming tool that must not be confused with error handling. Instead they should primarily be used to state something that due to a surrounding context is known to be true. I use `assert` this way to protect the code I write from its worst enemy, the maintenance programmer, which very often turns out to be, well, exactly: myself.

Validating function arguments is simply good programming practice and so is high cohesion. To simplify functions and increase their cohesion I often have them delegate to small, internal functions. When passing around pointers, I validate them once, pass them on and state the fact that I know they are valid with an assertion (please note that compilers prior to C99 take the macro argument to assert as an integral constant, which requires programmers to write `assert(NULL != myPointer);` for well-defined behaviour).

```
void sayHelloTo(const Address* aGoodFriend)
{
  if(aGoodFriend) {
    Telegram friendlyGreetings = {0};

    /* Add some kind words to the telegram,
    omitted here... */

    sendTelegram(&friendlyGreetings,
     aGoodFriend);
  }
  else {
    error("Empty address not allowed");
  }
}

static void sendTelegram
 (const Telegram* telegram,
 const Address* receiver)
{
  /* At this point we know that telegram
   points to a valid telegram... */
  assert(telegram);

  /* ...and the receiver points to a valid
   address. */
   assert(receiver);

  /* Code to process the telegram omitted...*/
}
```

In the example above `assert` is used as a protective mechanism decorated with comments describing the rationale for the assertions. As always with comments, the best ones are the ones you don't have to write because the code is already crystal clear. Specifying the intent of the code is a step towards such clearness and `assert` itself proves to be an excellent construct for that.

Applying the Idiom

The idiom ASSERTION CONTEXT increases the expressiveness of the code by merging the comment, describing the assertion context, with the assertion it refers to. This context is added as a string, which always evaluates to true in the assertion expression. The single-line assertion is now self describing in that it communicates its own context:

```
assert(receiver && "Is validated by the
 caller");
```

Besides its primary purpose, communicating to a human reader of this code that the receiver is valid, ASSERTION CONTEXT also simplifies debugging. As an assertion fires, it writes information about the particular call to the standard error file. The exact format of this information is implementation-defined, but it will include the text of the argument. With carefully-chosen descriptive strings in the assertions it becomes possible, at least for the author of the code, to make a qualified guess about the failure without even look at the source. As it provides rapid feedback, I found this feature particularly useful during short, incremental development cycles driven by unit-tests.

A drawback of ASSERTION CONTEXT is the memory used for the strings. In small embedded applications it may have a noticeable impact on the size of the executable. However, it is important to notice that `assert` is pure debug functionality and do not affect a release build; compiling with NDEBUG defined will remove all assertions together with the context strings.

CONSTANTS TO THE LEFT

Problem Context

The C language has a rich flora of operators. The language is also very liberal about their usage. How liberating the slogan "Trust the programmer" may feel, it also means less protection against errors and some errors are more frequent than others. It wouldn't be too wild a guess that virtually every C programmer in a moment of serious, head aching caffeine abstinence has erroneously written an assignment instead of a comparison.

```
int x = 0;

if(x = 0) {
   /* This will never be true! */
}
```

So, why not simply ban assignments in comparisons? Well, even if I personally avoid it, assignments in comparisons may sometimes actually make some sense.

```
Friend* aGoodFriend = NULL;
...

if(aGoodFriend = findFriendLivingAt(address))
{
   sayHelloTo(aGoodFriend);
}
else {
   printf("I am not your friend");
}
```

How is the compiler supposed to differentiate between the first, erroneous case and the second, correct case?

Applying the Idiom

By keeping CONSTANTS TO THE LEFT in comparisons the compiler will catch an erroneous assignment. A statement such as:

```
if(0 = x) {
}
```

is simply not valid C and the compiler is forced to issue a diagnostic. After correcting the if-statement, the code using this idiom looks like:

```
if(0 == x) {
  /* We'll get here if x is zero - correct! */
}
```

The idiom works equally well in assertions involving pointers.

```
assert(NULL == myNullPointer);
```

Despite its obvious advantage, the CONSTANTS TO THE LEFT idiom is not completely agreed upon. Many experienced programmers argue that it makes the code harder to read and that the compiler will warn for potentially erroneous assignments anyway. There is certainly some truth to this. I would just like to add that it is important to notice that a compiler is *not* required to warn for such usage. And as far as readability concerns, this idiom has been used for years and a C programmer has to know it anyway in order to understand code written by others.

Summary

The collection of idiomatic expressions in this article is by no means complete. There are many more idioms in C and each one of them is solving a different problem.

Idiomatic expressions at this level are really just above language constructs. Thus, the learning curve is flat. Changing coding style towards using them is a small step with, I believe, immediate benefits.

Acknowledgements

Many thanks to Drago Krznaric and André Saitzkoff for their feedback.

Adam Petersen

Uses Cases

Phran Ryder <phran@agilenorth.org.uk>

Requirements

Many organizations will be uncomfortable with the way requirements are gathered in Agile methodologies. For example, XP (eXtreme programming) uses small cards that can be prioritized and stuck to wipe boards. There are many reasons for this discomfort; tradition being, perhaps, the strongest. But there are practical reasons such as the personnel involved in a project being on multiple sites.

So what are the alternatives? The Rational Unified Process (RUP) defines requirements as Use Cases. But what are they? How do you write effective use cases? And can they be used in an agile way?

Alistair Cockburn

I have been writing use cases on and off for about a decade. During that time I never felt comfortable doing so, I never felt I was doing it well. When I discovered XP I was much happier writing requirements as user stories on cards. But I recently read the book "Writing Effective Use Cases" by Alistair Cockburn [1] and as a consequence I now feel I know how to write use cases. This article summarises Alistair's approach – but to get a real understanding, you need to read the book and do it.

The article also adds a little extension of my own that makes it easy to trace requirements.

I am not saying that use cases are the best way to write and manage requirements. I am saying that should you find yourself writing use cases, this is a very good way to do it.

Actors with Goals

Use cases define the requirements of a system; they define the desired behaviour of that system. They break down the description into a number of cases of when the system is used to achieve (or at least attempt to achieve) some **goal**. The use case name is some short description of that goal e.g. 'Set Up Profile'.

Each use case describes the activities involved in reaching that goal. The people or systems that can perform these activities are referred to as **actors**. For each use case there is a **primary actor** who has the goal.

When writing use cases, the word system is used for two purposes. We have the system that the requirements define, and the systems acting parts in the solution. Cockburn refers to the former as the **System under Discussion** (or SuD) and I will do too.

So that seems simple, actors with goals. Shortly I will go into more detail but before then I want to dispel a myth.

Not Use Case Diagrams

Please, please, please note well, understand, and absorb…a use case is not the same as a use case diagram. A use case diagram contains stick men (representing actors) connected to bubbles that contain text describing a goal that the actors are involved in reaching.

If you have several use cases, use case diagrams are useful in that they can:

1. Show the relationships between the actors and several use cases
2. Provide an overall view of the desired system behaviour
3. Illustrate the context and boundaries of the system.

But that's it. Beyond that they provide little value. The true, deep, useful value comes from the text in the use cases. This article summarises how to write and structure the text so that it provides real value to a project.

Too many people start requirements gathering by drawing stick men. But that is the wrong thing to do. Use case diagrams are useful for high level information, but you can easily get by without them, and they are certainly not the place to start. I'll not mention them again.

Value in the Words – and Their Structure

The table on the right is my current preferred way of structuring use cases. Cockburn suggests others and suggests you find one that suits you.

The left hand column contains names for **use case parts**. The part of initial interest is the Main Success Scenario. These contain the steps that are completed in order to reach the goal. Cockburn recommends that you write these as simple, single sentences of the form:

Subject…verb…direct object…prepositional phrase

For example:

Any User logs onto the SuD and adds the user with an empty profile.

The main success scenario describes one set of actions in the SuD. Of course there are always alternatives.

Alistair's Trousers

The main success scenario is likely to be one of a number of scenarios that can take place when trying to reach a goal. A use case is in reality a collection of scenarios. The alternative scenarios are recorded as extensions to the main scenario. If something different can happen in an action step, the extension records the condition that leads to the alternative followed by the steps that would then take place for that scenario.

Some of these extensions are success scenarios that result in the goal being reached, while some are failure scenarios that result in the goal not being reached (thus the action steps might be recovery actions).

Amusingly, Cockburn views these as a pair of stripy trousers. Each strip is a scenario with success scenarios go down one leg and failure scenarios go down the other leg.

Stakeholders with Interests

Each system will have a number of stakeholders. When building the system it is essential to ensure the interests of ALL the stakeholders are met. And a system meeting those interests will be produced sooner if their interests are considered early. It is too easy to think of just the actors, their goals and the business requirements. Considering the interests of all stakeholders helps to ensure the requirements are thorough and complete.

Related to the scenarios and stakeholder interests are the following use case parts:

1. Preconditions: what we expect the state of the world to be
2. Minimal Guarantees: the interests protected on any exit
3. Success Guarantees: the interests satisfied on a successful ending
4. Trigger and Frequency: the action that starts the use case and how often it occurs

Use Case #2	**Set Up New Profile**
Context of Use	A new user, along with the access they have, is added to the SuD.
Scope	SuD
Level	Primary Task
Primary Actor	Administrator
Stakeholders and Interests	Managers: Some text describing their interests Users: ditto Audit: ditto Security: ditto
Preconditions	Required authority level has been provided for the primary actor.
Minimal Guarantees	Changes to profiles are logged.
Success Guarantees	User's profile is modified.
Trigger and Frequency	Ad Hoc.
Main Success Scenario	
1.	HR register user with Active Directory
2.	Any User logs onto the SuD and adds the user with an empty profile.
3.	SuD Adds Change To Access Profile Log recording the addition of a new profile.
4.	Administrator logs on to the SuD and Modifies Profile
Extensions	
1	User is already registered with Active Directory The registration system prevents re-registration
2	User to be added already has a profile The SuD prevents re-adding User to be added is not registered with Active Directory The SuD prevents addition of user.
2-3	SuD cannot be accessed SuD will be accessed another time
Open Issues	
1	Can some of the data be derived

Onions have Layers – Ogres have Layers

And so do use cases.

Requirements aren't really defined or written. They evolve. They evolve as the understanding of the problem area increases and as that problem area changes. An important principle in software development is to concentrate on activities that provide best value to the business. This is true when writing requirements as use cases. Cockburn refers to this as 'Managing your energy'.

In the initial stages of requirements gathering, we are not interested in detail, in alternatives, in error conditions, in issues. We want a high level understanding of the problem area. For this Cockburn suggest getting started with **usage narratives** – a situated example of the use case in operation. This is a good place to start as you are creating a first set of use cases that scope the SuD.

As you add detail you can move onto **Use Case Briefs**. These are two to six sentence descriptions mentioning only the most significant success and failure scenarios.

As understanding of the requirements increases, additional use case parts can be added (Stake holders, guarantees etc.) along with more detail and additional scenarios.

A further tool for managing energy, and concentrating on providing value, is the **open issues** use case part. This can be used to record scenarios, problems, risks etc. that you don't want to forget about but don't want to look into at that point.

As the use cases grow, they may come a point when a use cases will warrant being split and further new use cases will be spawned that go into greater detailed. When this happens we need a mechanism for referencing use cases that 'call' each other to provide the detail. This is done simply by underlining the text of the used use case. In table 1, Modifies Profile is a reference to another use case.

Cockburn uses named goal levels to help manage the granularity of use cases. The main levels are:

1. User goals: define elementary business processes.
2. Summary Level: link several uses cases providing an overview and context.
3. Sub functions: these are goals required to carry out the user goals and often appear in several User goals

As the requirements evolve it is often the case that a use case no longer warrants existing on its own so it is merged back into a higher level. At other times similar use cases will be found that can be merged into one that has parameters. This is merging and splitting of use cases is of course akin to re-factoring of code. Because of this well written use cases can be very robust to change and, to a certain extent, re-usable.

Agile developers might argue that all this requirements defining activity hasn't actually produced anything. Certainly no code has been produced but there is no point in producing code if it is not the most valuable thing to do. Producing use cases might be the most valuable thing to do in large organizations or on large projects that are likely to cost several million. In this situation the most valuable thing for business is to understand what they really want and need, along with a good idea of the costs, risks, benefits and alternatives.

Having said that, when we view the use cases as a collection of scenarios (stripes on the trousers) we can see how use cases can be used in an agile development process. As we manage our energy and concentrate on providing value, we start by defining the main success scenarios. These tracer bullet [2] scenarios are the most important and we could implement them at this point – if it is valuable to do so.

Phran's Addition

The use case template I currently use places the use case part in a table. The reason for this is that it makes it easy to trace the use cases back to high level requirements and/or forward to estimates or implementation, unit tests or whatever.

The amount of tracing you do depends on the process you have. CMM generated processes seem to require high levels of tracing while Agile processes keep tracing to a minimum. The approach described here allows you to e.g.

1. Show that higher level requirements are being met
2. Show that an estimate covers all the requirements
3. Easily see how removing or adding requirements affects the estimate
4. To trace the functional tests to the requirements thus proving that the solution satisfies the requirements.

This is done simply by introducing further columns to the right (see below).

Conclusion

Use cases written in this way have many advantages:

1. They make it easy to manage your energy and allow you to focus on providing value to the business.
2. They focus on the interests of stakeholders.
3. They can evolve iteratively and incrementally, and provide early opportunities for implementing early.

I will say again: I am not saying that use cases are the best way to write and manage requirements. I am saying that should you find yourself writing use cases this is a very good way to do it.

Phran Ryder

References

1 Cockburn, Alistair (2001) *Writing Effective Uses Cases* Addison-Wesley ISBN 0 201 70225 8

2 Hunt, Andrew and David Thomas (2000) *The Pragmatic Programmer* Addison-Wesley ISBN: 0 201 61622 X

Phran Ryder is Chairman of AgileNorth.org.uk – a non profit organisation for technical and business staff who wish to learn and share experience of becoming and being agile – details at: `www.agilenorth.org.uk`.

Use Case #2	**Set Up New Profile**	**Tracing**
Context of Use	A new user, along with the access they have, is added to the SuD.	
Scope	SuD	
Level	Primary Task	
Primary Actor	Administrator	
Stakeholders and Interests	Managers: Some text describing their interests Users: ditto Audit: ditto Security: ditto	
Preconditions	Required authority level has been provided for the primary actor.	
Minimal Guarantees	Changes to profiles are logged.	
Success Guarantees	User's profile is modified.	
Trigger and Frequency	Ad Hoc.	
Main Success Scenario		
1.	HR register user with Active Directory	2.1
2.	Any User logs onto the SuD and adds the user with an empty profile.	1.1, 3.3
3.	SuD Adds Change To Access Profile Log recording the addition of a new profile.	4.5
4.	Administrator logs on to the SuD and Modifies Profile	4.6, 4.7 4.8
Extensions		
1	1. User is already registered with Active Directory a. The registration system prevents re-registration	
2	1. User to be added already has a profile a. The SuD prevents re-adding 2. User to be added is not registered with Active Directory a. The SuD prevents addition of user.	2.2
2-3	1. SuD cannot be accessed a. SuD will be accessed another time	
Open Issues		
1	Can some of the data be derived	

AI – Expert Systems

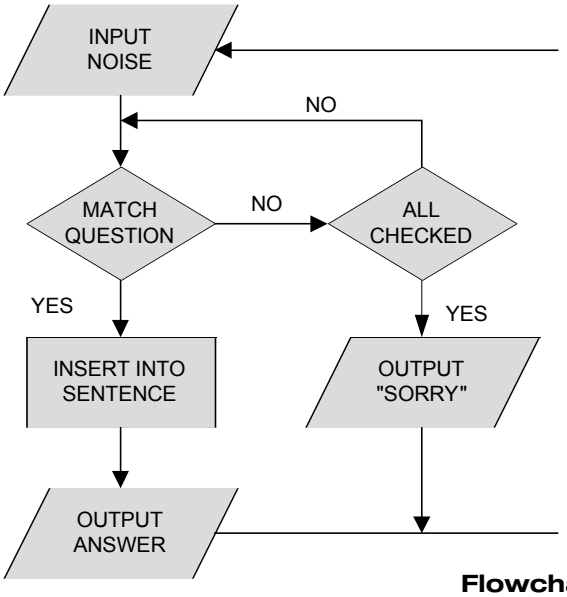
Steve Hopley <shopley@sthelens.ac.uk>

A simple definition of a human expert is a *person who knows a great deal about a subject and who can give sensible advice on it.* It takes ages to attain this level of knowledge, so experts are few and far between and as such are rather like the proverbial Time Lord – not always there when you need them.

Due to this lack of experts, people decided to teach computers to become experts. These experts are always on hand, never need to rest, don't go on strike and best of all, really do know what they're talking about. The problem is that if you ask them something on a subject they're not trained in, they're as dumb as a sack of rocks! Why? Computers have to be programmed step wise and logically.

For programmers to understand how to create an "expert system", they needed to look at how an expert works. It would be pointless to take a very complex problem, so the simplest of problems known was used (it doesn't matter what the problem is either – the following fits anything).

1. Take in the available information on the current task
2. Compares this information with previously stored information and looks for a match
3. Reports if there is a match and acts on it.



Flowchart 1

For this, all that is required is nothing more than a simple database – the sort of thing which could be written in MySQL in a couple of minutes. Great for those who can use SQL, useless for humans as humans don't speak SQL. What would be required is a system which would accept entry in the speakers native language (flowchart 1).

(In reality, an actual expert system will have a user interface and the knowledge engine in the background – this can be a database or commonly, something written in LISP)

To keep things simple, for now, I'll stick to a fixed input format. So that you can visualise the problem in hand, we will look at recognising animals by the sound they make.

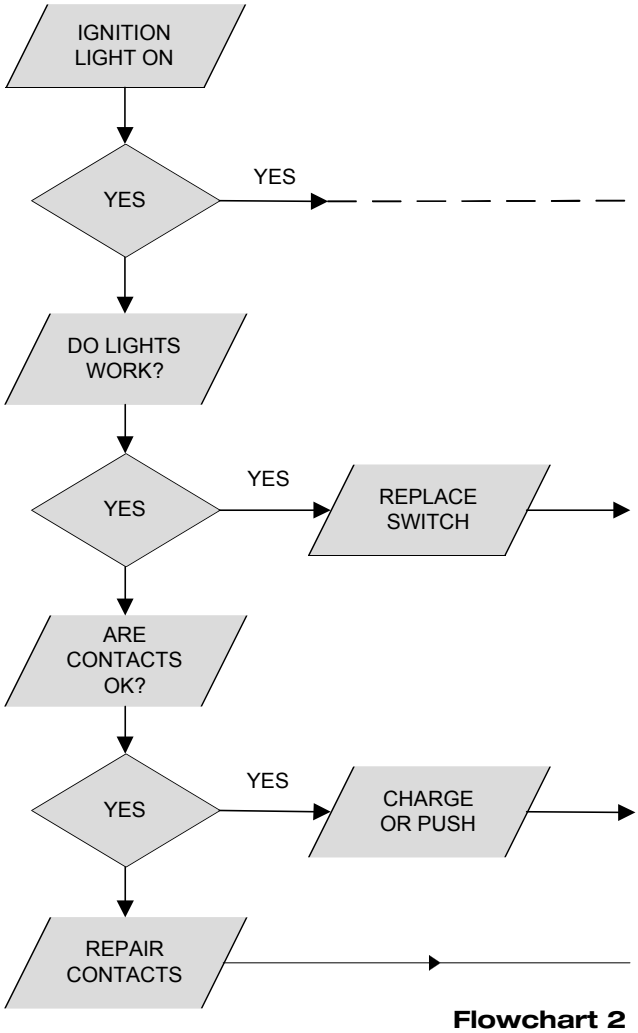
For this, it is simple enough to create two arrays of data, `q` (for the sounds) and `a` (for the animals). This is then followed by a simple comparison between the entry (sound) and the contents of `q` in turn. This is a trivial matter to create and I will leave it up to the reader to devise such a program.

Branching

The above example is a very simple one: there is only one question and one answer. In reality, far more complex problems are posed and until a series of questions have been asked, an answer cannot be found.

A case in point is a broken down car. You have placed the key in the ignition, turned it and ... click! Question is, why has this happened? There are a number of possible reasons for this:

- Flat battery
- Bad connections
- Switch broken



Flowchart 2

- Starter jammed
- Starter broken
- Solenoid broken

To find the cause, a process of elimination is used.

- Is the ignition light on?
  If it isn't, then there is no power at the switch, so any of the top three possibilities are likely. This can be narrowed down further.
- Do the lights work correctly?
  If they do, then obviously the battery is fine and it must be correctly connected to the light switch. It looks like the switch is duff. Whip it out and replace. If the lights don't work, check the connections.
- Are the battery connections OK?
  If they are, the battery is flat and you need to push it.

A sequence of checks could be made to deal with a situation where there is power but no starter mechanism (the last 3 possibilities).

The simplest way to program this is using a branching structure using a series of IF / THEN tests (see left). Again, not rocket science to program (so I won't bore you with it here!).

This is, of course, a very cumbersome method. As more choices are given, the lack of efficiency (in terms of application running) of this top down method becomes apparent.

Pointers

Why bother with such an inefficient method when a far more flexible (and efficient) method is to place the text into arrays and use a pointer to direct to the next question or reply, according to if "yes" or "no" has been entered by the operator to the current question (flowchart 3).

The format for entering the data for each branch point is

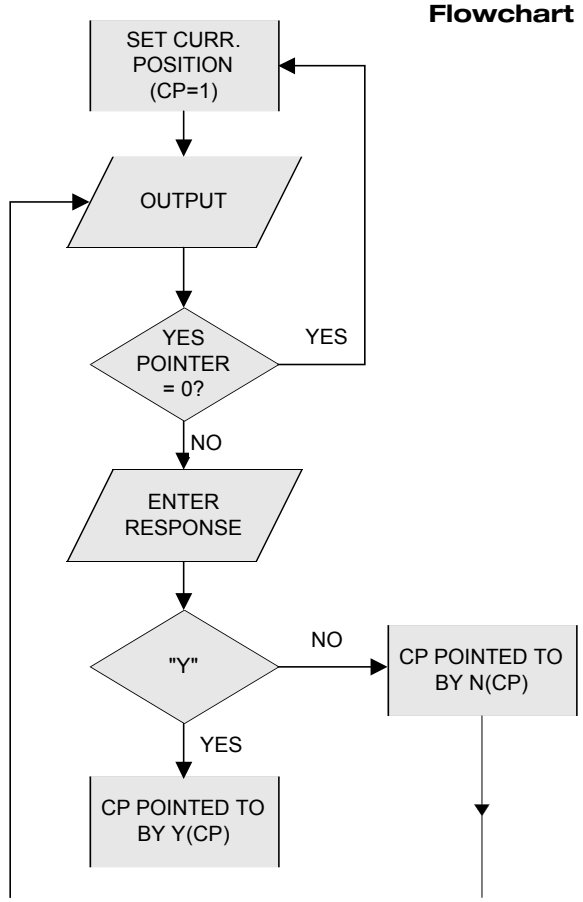
```
(TEXT), (Pointer for "Yes"), (Pointer for "No")
```

The first question was

Is the ignition light on? (Y/N) ... 1

If the answer was "N", then the second question is asked

Do the lights work correctly? (Y/N) ... 2



Flowchart 3

Otherwise you continue with the other part of the diagnosis.

For this to work, 3 arrays are used; one for the output text and two for the pointers to the next answer (`y` and `n`). To make life simple, an `int` variable, `np`, is used for the number of questions available (in this case, 7). Here is where the programming becomes slightly more sticky. You need more of a linked list with the following structure

```
char text[]
pointer->yes
pointer->no
```

It's not quite a linked list as there isn't a pointer to the next question (in the list) as the next question depends on the answer. The two pointers can either be the array index or real pointers in a list. For example, in pseudo code terms it may look like this

```
char question[][1] = "Do the lights work
 correctly ? (Y/N)";
int yes = 3;
int no = 4;
```

The running routine for this procedure is very simple. We set up a pointer for the current position, `cp`, which is initially set to `1` (you'll see why in a moment) and the first text outputted. The end of the routine is when `yes[cp] = 0` (very unlikely if `cp = 1`!). At the end of the run routine, `cp` is reset to `1`. Failing that case, the next `text[cp]` (where `cp` equals either the `yes` or `no` pointers).

Of course, this approach, while it is becoming far more efficient in terms of code and logic, still requires a very step wise approach. Nothing wrong in that, but as with the basic "expert" system, unless the problem is trivial, the code can become very hairy! As an alternative to the systematic approach, another is available.

Parallel Approach

The parallel approach differs from the sequential approach in that it asks all of the possible questions before it reaches its conclusions. The method takes longer than following the efficient tree structure, but is more likely to produce a correct answer as no points of comparison are omitted.

Using a comparison of forms of transport, this becomes easier to understand. I'll consider eight features and mark them with a `1` or `0` for the presence or absence of these in each of our five modes of transport (see the table below). The pattern in the results shows that because of the variety, it must be possible to decide which mode of transport has been selected.

	Bicycle	Car	Train	Plane	Horse
Wheels	1	1	1	1	0
Wings	0	0	0	1	0
Engine	0	1	1	1	0
Tyres	1	1	0	1	0
Rails	0	0	1	0	0
Windows	0	1	1	1	0
Chain	1	0	0	0	0
Steering	1	1	0	1	1

As before, we can set up as 2 arrays; one for each of the vehicles and one for the choices (though again, a linked list would probably be a far more efficient method of storage – for the purposes of this piece, I'll keep to arrays). A simple driver program starts with the first question (does it have wheels?). If the answer is yes, then instead of another specific question being asked, all of the available vehicles with wheels are given (flowchart 4).

If "Y" was answered to the wheels question, then the expected output would have been bicycle, car, train and plane. However, if we had said "N", then only horse would have been given.

This does though demonstrate a problem with the parallel approach. Even though the horse was correctly identified as being the only mode without wheels, the program still insisted on asking each question before it would decide on the correct answer.

While on the surface that seems wasteful, if you answered yes to the next question (does it have wings?), the machine will refuse to believe in flying horses.

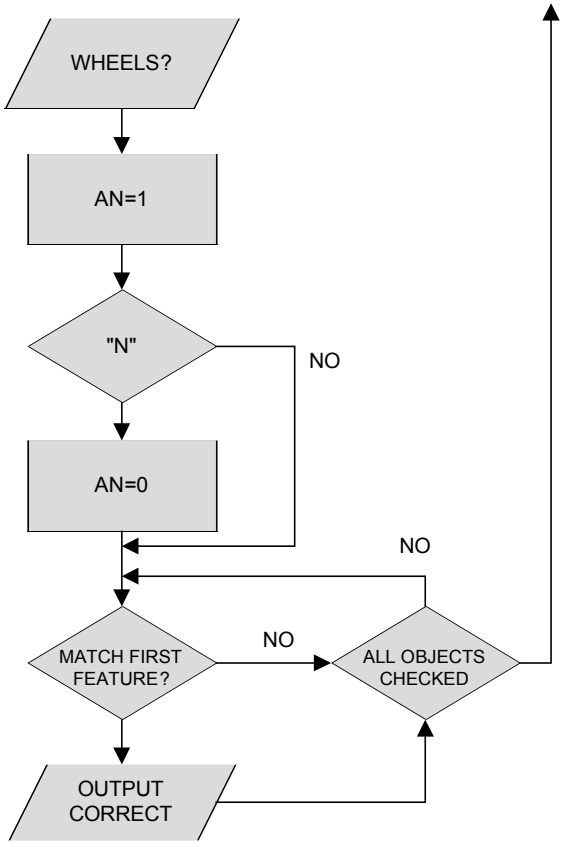
In code terms we would have something like figure 1 (overleaf).

By having the questions as a separate function, we can test for all of the questions in turn (see flowchart 5 on following page). We simply change the `if (truth[c][0] ...)` to read `if (truth[c][questions] ...)`.

This is now starting to resemble a form of an expert system. The machine can now respond correctly to identify a limited set of transport modes based on a fixed series of questions.

The problem with the above approach is that it prints out a list of matches for each question asked, but isn't actually going to give an authoritative result based on which data set gave the best overall match.

That is fixed by adding a score to the routine. Again, this is in the form



Flowchart 4

```
#define MODES 5

char vehicles[][MODES]={"Bicycle", "Car",
 "Tram", "Plane", "Horse"};
bool truth[MODES][7]={1, 0, 0, 1, 0, 0,
 1, 1},{/*other answers */};
char questions[][7]={"Does it have wheels",
 "Does it have wings", /* other questions */};

void questions(int question)
{
   char answer;

   int an = 1, c;
   printf("%s (Y/N) ? ", questions[question]);
   scanf("%s", answer);
   if (answer == 'N')
     an = 0;
   for (c = 0, c < MODES, ++c)
     {
        if (truth[c][0] == an)
          printf("%s\n", vehicles[c]);
     }
}
```

Figure 1

of an `int` array which is incremented when the match is found for that particular dataset (flowchart 6).

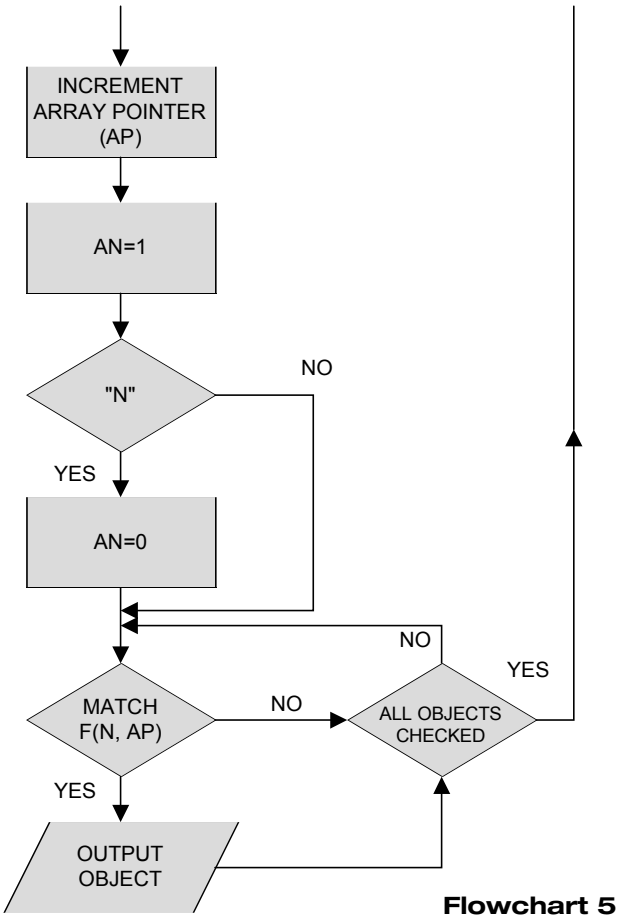
If a complete match is found then `S(N)` will be 8 with anything less meaning that matches were found, but it may not be the correct answer.

The problem is, what actually is the right answer. It can be argued that the question "Is it steerable" may not apply to a horse. It's a matter of opinion.

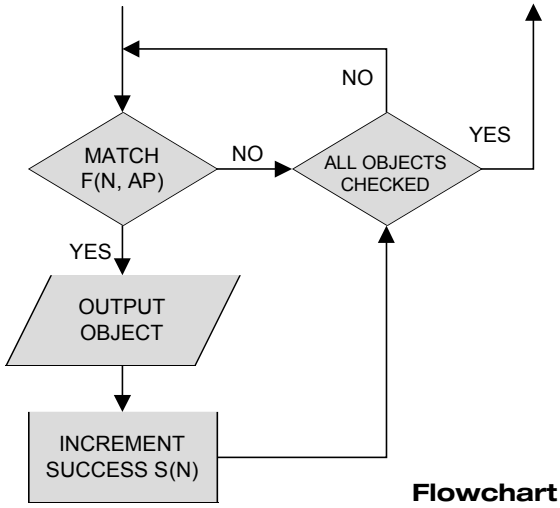
By using the scoring method, this sort of problem doesn't matter as the highest value of `S(N)` is probably going to be the right answer anyway – the only caveat to this is that the answers are all equally weighted.

Can this system be improved upon and if so, how?

We can optimise the code as there are 8 questions and assuming a byte length of 8 bits, it means we can use binary to store the truth table (opposite). This means that instead of having a number of large arrays, we have a single char which contains all of the truth table for each choice.



Flowchart 5



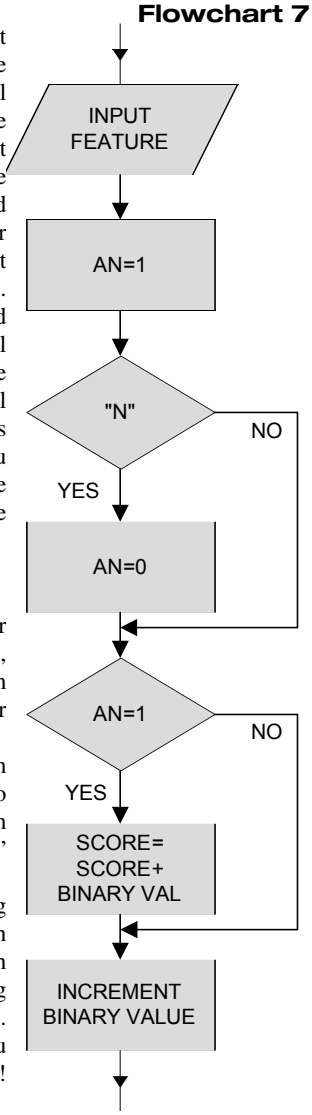
Flowchart 6

A simple bitwise test (`if (x & position) ...`) is then performed with the resulting decimal number being placed into the scoring array (flowchart 7).

You should have spotted something by now. We only need to keep track of the total number produced `(S)` by adding the binary values of the yes answers; we have no need to continually loop through and check each part of the array contents time. The only information that we need to provide to the program is the decimal value for a 100% correct answer and when all the questions have been asked, check against the decimal values (which we found by converting the binary to decimal – flowchart 8 on next page). Suddenly, we're starting to get code which is efficient, fast, more flexible and becoming more of an expert. Remember though, binary only works of expert systems that just perform yes/no questions.

	Bicycle	Car	Train	Plane	Horse
Wheels	1	1	1	1	0
Wings	0	0	0	2	0
Engine	0	4	4	4	0
Tyres	8	8	0	8	0
Rails	0	0	16	0	0
Windows	0	32	32	32	0
Chain	64	0	0	0	0
Steering	128	128	0	128	128
Sum total	**201**	**173**	**53**	**175**	**128**

Flowchart 7



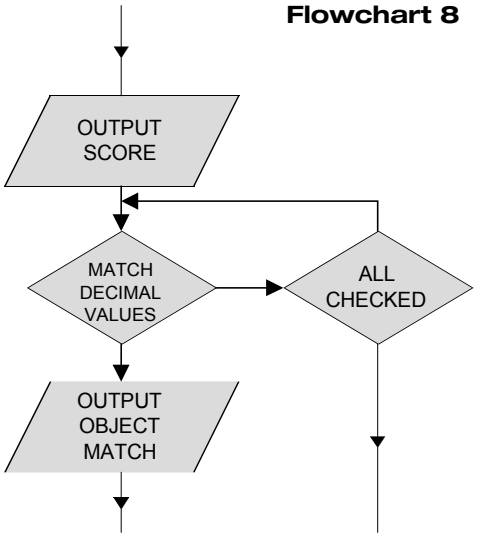
We have saved a lot of memory and time as each array only takes a small number of bytes. The downside is that we now need this binary to decimal conversion, which also gives no clues as to when a complete match is found (you cannot take the nearest decimal value here as the the value depends on the position).

Dynamic Learning Systems

Now that you have the basics under your belt and hopefully have followed this, let's move this up a notch and see if from a very simple expert, we can make our expert think for itself.

This is not as difficult as it may seem either – we just have to figure out how to move from a rigid "rules" based system to one whereby the program "learns" from its mistakes.

The key to this transition is enabling the program to work out the decision rules for itself and provided the human tells the machine when it has gone wrong (though not where it has happened). Why? Well there is nothing to say you know what the rules are exactly yourself!



Flowchart 8

Let us start out with a series of features which enable us to distinguish between the different objects, but without the predefined yes/no pattern (or the decision rule) to guide. Instead, we allow the program itself to calculate what the pattern should be.

If we stick with the transport example, we have a familiar basis. This time though, we alter the program so that we have some new variables. `FE` is the number of features (in our case, 8), a char array, `F`, which contains the names of the features, an `int` array, `FN` which holds the values you give to each feature as input at any given point (`0` or `1`) and a final `int` array, `R`, which will hold the current overall values of the decision rule of the feature.

```
#define FE 8
char F[][FE] = {"Wheels", "Wings", "Engine",
 "Tyres", "Rails", "Windows", "Chain",
 "Steering"};
int FN[FE], R[FE];
```

Now, consider each feature in turn (flowchart 9) as it requires some explaining. First the current feature value is set at `0` for the initial cycle and the first input is requested. If the response back is "Y", then the value element `FN(N)` is set to `1`. This produces the pattern which describes the object in the `FN` array.

Next, the decision variable, `DE`, is set to zero. It is then recalculated as the sum of the current value of `DE`, plus each of the feature values entered, `FN(N)`, multiplied by the current decision rule, `R(N)`.

Nothing hard there. But there is a fly in this ointment.

Which is Which?

Let's consider the simplest situation where there are only two possibilities – a bicycle or a car. Initially, the distinction between them is fairly arbitrary by saying that if the final value of `DE >= 0`, then it's a bicycle – anything less than `0` and it's a car. It doesn't actually matter that this is not really true as the system will correct itself. When the program has made a decision based on the value of `DE`, confirmation of the result is asked for.

There are three possible courses of action according to if the decision was correct – each course results a weighting, `WT`.

1. If the decision is correct, effectively no action is taken (`WT = 0`). The program loops back for another go.
2. If `DE>=0` and the computer was wrong, then `WT = -1`
3. If `DE<0` and the computer was wrong, the `WT = +1`

The effect of `WT` is to modify the values in the rule array, pulling them down if they are too high and pulling them up if they're too low. This can be demonstrated thus. The program is written and the following data entered.

Wheels	Y	Wings	N
Engine	N	Tyres	Y
Rails	N	Windows	N
Chain	Y	Steering	Y

The program returns with `DE = 0` as this is the initial value and no modifications have yet taken place. As `DE = 0`, the system assumes that the object is a bicycle and asks for confirmation. As it is a bike, the confirmation is "Y". If you were to look at the `R` array, all of the values will still be zero as nothing has changed.

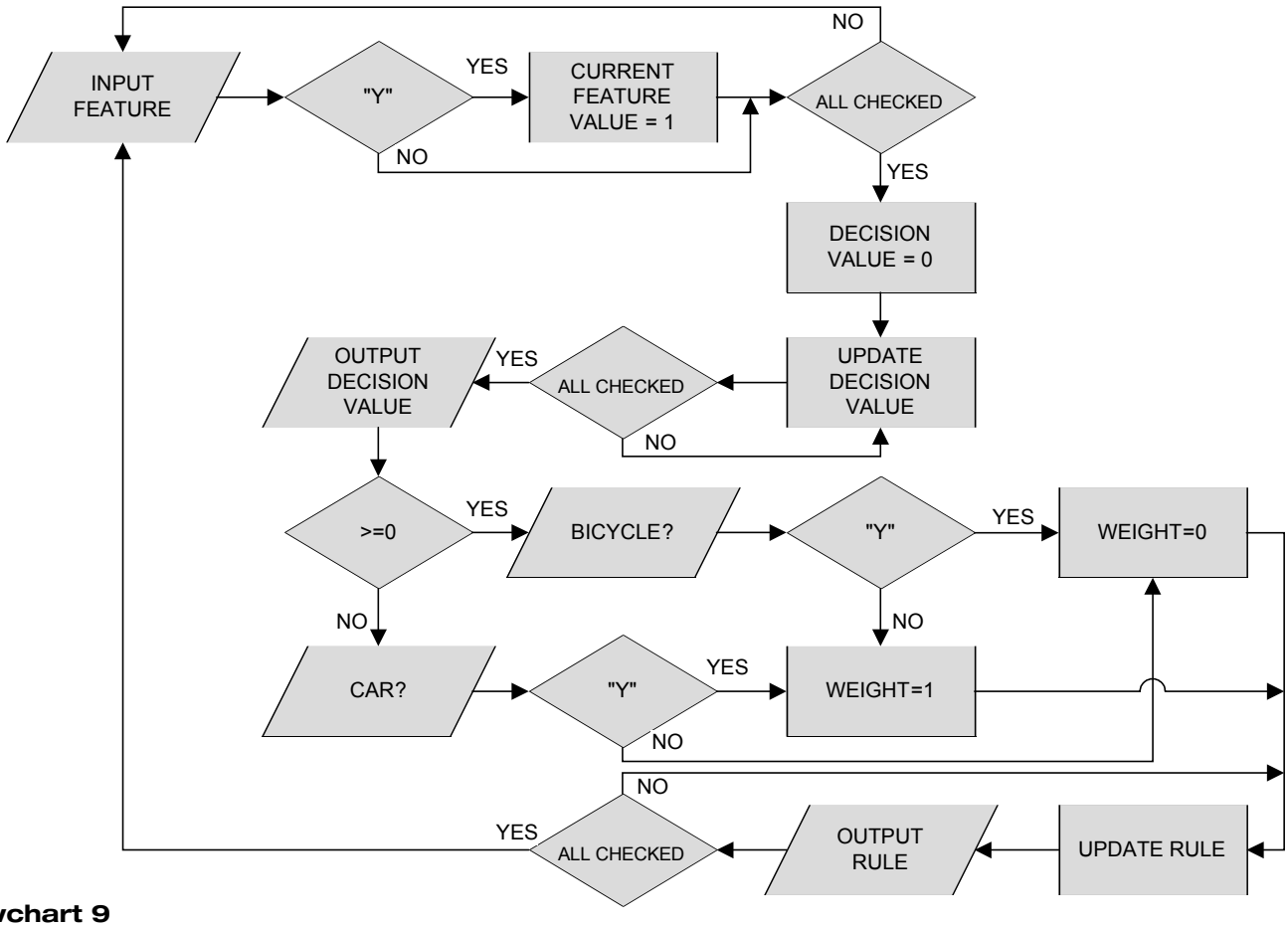
Next, the following is entered for a car.

Wheels	Y	Wings	N
Engine	Y	Tyres	Y
Rails	N	Windows	Y
Chain	N	Steering	Y

`DE` is still `0`, so the wrong conclusion is reached and the answer when confirmation is asked about if it is a bicycle is now "N". As a mistake as been made, the decision rule is modified by subtracting one from each value in `R` where a "Y" answer was given. The contents of the rule array will therefore now be as shown on the right.

Wheels	-1	Wings	0
Engine	-1	Tyres	-1
Rails	0	Windows	-1
Chain	0	Steering	-1

If you now were to re-enter the values which describe a car, the program comes up with the correct answer (Is it a car?). `DE = -5`. Okay, the program is giving the correct answer for a car, but if you enter the values for a bicycle (just as a check),



Flowchart 9

the machine gives the wrong answer. Why? Let's look at the input and the array `R`

Wheels	Y	0	Wings	N	0
Engine	N	-1	Tyres	Y	0
Rails	N	0	Windows	N	-1
Chain	Y	1	Steering	Y	0

`DE = -3`, as it is less than 0 the computer thinks that it is a car. All is not lost though. The positive features which are common to the car and bike are now increased by one, so that if you repeat the sequence, the correct answer is given (`DE = 1`). The machine can now correctly distinguish between a car and a bicycle.

Again, a test is performed for a car and the final result of `DE = -2` is given. If you look at the rule array values, you'll see that these correspond in both the number and position to the unique features which distinguish these objects (*Chain for bicycle, Engine and Windows for car*).

We can see that the machine has learned by its mistakes and is making a decision as to what an object is for itself. Not exactly earth shattering though. Any 6 month old kid can do that – but that's part of the problem with anything to do with AI, we're comparing it to too high a life form. At best, AI is at the 100 – 1000 braincell level (a slug is a good example), but we've made a start.

Let's see if we can expand on this system to deal with a wider number of possibilities (flowchart 10). To start with, let's define the number of objects we want to recognise, `OB` and place those objects as before into a `char` array. Next, set up a decision rule array, `R[FE][OB]` which holds the rules for each object and finally a decision array, `D(n)`, to hold the decision values for each object.

The process for the program required follows this form

1. All of the decision variables are initialised to `0`.
2. The values for each feature are entered in the same way as the examples above.
3. Each element of `D(n)` is updated according to the status of the entered values `FN(n)` and the contents of the appropriate rule element, `R[n][m]`.
4. We need to look to see if any of the decision values for any of the objects `D(n)>=DE`. If this is true, set a top score variable, `TS`, equal to the number of the object producing the best match, `N`.
5. As the best guess of the system is that this is the correct answer, the machine asks for confirmation and returns for a new input without making any changes if the answer was correct.
6. If it is the incorrect answer, the names and numbers of all of the objects are outputted and the user is asked for the number of the correct answer, `CR`. (The program limits the values of `CR` to prevent crashes.
7. The next check is whether the decision value of each object, `D(n)`, is greater than or equal to the overall decision value, `DE` and whether the object being considered is not the correct answer. If both are these are true, update the rules by subtracting the overall feature values, `FN(n)`, to bias favour of the correct answer.
8. The correct feature values for `FN(N)` are added to the rule array for the correct object to bias in the opposite direction
9. At the end, the rules are outputted.

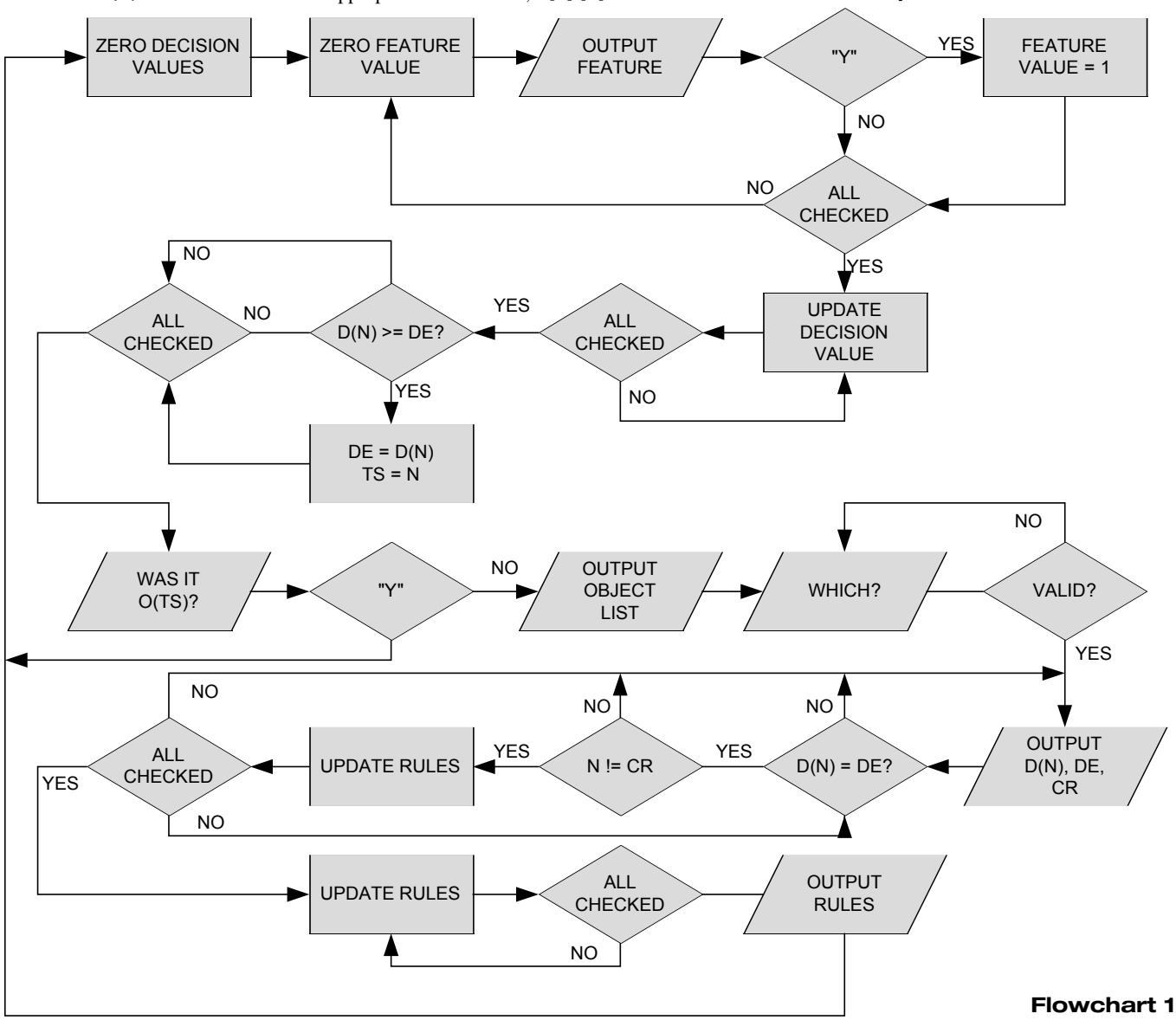
To demonstrate this, consider what would happen with the following input:

Wheels	Y	Wings	N
Engine	N	Tyres	Y
Rails	N	Windows	N
Chain	Y	Steering	Y

The program concludes that this is horse, so you tell it (via the keyboard) that the conclusion is incorrect. Next the program asks if it is a bicycle, you say "Y" as it is.

```
Was it a horse?      N
1    Bicycle
2    Car
3    Train
4    Plane
5    Horse
Which was it?        1
```

[concluded at foot of next page]



Flowchart 10

F2C – Is it a Practical Solution?

Derek M. Bloor, University of Salford

Derek was a colleague of mine during my time within the School of Chemistry at the University of Salford. At some point during our final year there, we began investigating if there was any value in using f2c as an alternative to re-writing all of our research source code directly into C (the reason being that the F77 compiler we had been using was no longer up to the job and as the project was unfunded, we could not move over to another product).

In January 2003, Derek sadly passed away – many years before his time should have been. The work was mostly finished by the time we both left. What is presented here is the majority of the work covered with my notes completing the picture. I have removed some of the more Chemistry based aspects (very complex maths).

I must thank Derek's widow for her help in this and allowing me access to his notes.

Paul Johnson

Rationale

FORTRAN is a simple mathematical computer language designed for high level maths processing and calculation manipulation. It is ideally placed for many Physical Chemistry based applications where number crunching and manipulation forms an integral aspect of the design and primary investigation basis. It is an invaluable tool in the prediction of behaviour for given parameters.

FORTRAN however is not a simple language to program in and therefore when factoring in the manpower costs for a research proposal, this time has to be taken into account. A simple FORTRAN program may take less than 200 lines of code, yet because of the specialisation required to use both the language and the compiler, a C program may be cheaper despite being many times longer (and potentially slower) due to the availability of C programmers.

The problem arises though with highly specialised code currently running and written in either pure or a variant of FORTRAN (for ease, FORTRAN will from now be known as F77) and what to do with it. To analyse this, three points have to be taken into consideration

[continued from previous page]

The status of the various decision and rule arrays are now given (right and below) for your information

You should be able to see that the features which have caused alterations in the rule arrays are wheels, tyres, chain and steering (which are all defined as bicycle features, but which aren't found in horses). Additionally, that all the values for the bicycle are `+1`, with everything else either `0` or `-1`.

Next, the features for the car are given. As the machine doesn't have a clue what a car is, only the values it has learned, it mistakes the car for a bicycle which the user then corrects. Note that the rule arrays for the bicycle and car (below) are amended to take into account the provided information.

D(N)	DE	CR
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1

Wheels	Y	Wings	N
Engine	Y	Tyres	Y
Rails	N	Windows	Y
Chain	N	Steering	Y

Wheels	Wings	Engine	Tyres	Rails	Windows	Chain	Steering	
1	0	0	1	0	0	1	1	**Bicycle**
-1	0	0	-1	0	0	-1	-1	**Car**
-1	0	0	-1	0	0	-1	-1	**Train**
-1	0	0	-1	0	0	-1	-1	**Plane**
-1	0	0	-1	0	0	-1	-1	**Horse**

```
Was it a bicycle? N
1 Bicycle
2 Car
3 Train
4 Plane
5 Horse
Which was it? 2
```

D(N)	DE	CR
3	3	2
-3	-3	2
-3	-3	2
-3	-3	2
-3	-3	2

Wheels	Wings	Engine	Tyres	Rails	Windows	Chain	Steering	
0	0	-1	0	0	-1	1	0	**Bicycle**
0	0	1	0	0	1	-1	0	**Car**
-1	0	0	-1	0	0	-1	-1	**Train**
-1	0	0	-1	0	0	-1	-1	**Plane**
-1	0	0	-1	0	0	-1	-1	**Horse**

This is repeated for a plane. Again, the machine has no idea what a plane is and guesses that it is a car and is then corrected.

```
Was it a car? N
1 Bicycle
2 Car
3 Train
4 Plane
5 Horse
```

Wheels	Y	Wings	N
Engine	Y	Tyres	Y
Rails	N	Windows	Y
Chain	N	Steering	Y

```
Which was it? 4
```

The rules are corrected and so we decide to give a train. Which for some reason best known to the machine, it thinks is a plane!

Wheels	Y	Wings	N
Engine	Y	Tyres	Y
Rails	N	Windows	Y
Chain	N	Steering	Y

```
Was it a plane? N
1 Bicycle
2 Car
3 Train
4 Plane
5 Horse
Which was it? 3
```

Finally, the last object is a horse. Surely, it must get this one right – I mean, what else can it be other than a horse? You guessed it – a plane!

```
Was it a plane? N
1 Bicycle
2 Car
3 Train
4 Plane
5 Horse
Which was it? 5
```

Wheels	Y	Wings	N
Engine	Y	Tyres	Y
Rails	N	Windows	Y
Chain	N	Steering	Y

This will continue for as long as you wish to enter data and eventually, the expert system will have learned enough to give the correct answer every time. How long it takes depends on the number of differences between the objects and on the order the object are presented to the expert. It can take a long time before the system becomes a true expert.

The final state of our 5 object system in the decision array shows the following (at the foot of the page) – and given the range of values, you can surmise how long it took to reach.

This has been a very simplistic demonstration on how an expert program may learn by its mistakes. In reality, masses of data would be automatically fed into the system and left for a long time to generate the correct answers. However, the approach taken here does mean that via a distributed network, many machines can plough through the data and arrive at an expert system.

Steve Hopley

Wheels	Wings	Engine	Tyres	Rails	Windows	Chain	Steering	
1	0	-1	1	0	-2	3	0	**Bicycle**
-1	4	1	0	-1	1	-2	0	**Car**
0	-1	1	-2	2	1	-1	-2	**Train**
-2	6	0	-1	-1	0	-2	-2	**Plane**
-1	0	0	-1	0	0	-1	0	**Horse**

1. Would the code generated using a package called f2c be a viable alternative in terms of maintainability, readability and speed of operation?
2. The length of time (and therefore cost to the project) required to rewrite the F77 code by someone competent in both F77 and C
3. The age old maxim, if it's not broken – why fix it?

Analysis

For the analysis of this, I am using two small applications and one large application, all three currently written in standard F77. The applications are

- A linear regression analysis program
- An elliptic integral solver
- MOPAC 6

The level of complexity rises as you proceed down the list, with MOPAC being a highly specialised molecular orbital package (with a source archive of around 1Mb).
For the purposes of this demonstration, only the linear regression source will be considered.

Linear Regression Source Code – F77

```
C Linear regression utility
      PROGRAM LINREG
      COMMON/SETUP/X,Y,N

COMMON/SIGMAS/SIGMAX,SIGMAX2,SIGMAY,SIGMAX22,A
VEX,AVEY

COMMON/BIGGIES/XLINE,YLINE,GRADIENT,STDDEV,R2
      DOUBLE PRECISION
SIGMAX,SIGMAX2,SIGMAY,SIGMAX22,AVEX,AVEY
      DOUBLE PRECISION
XLINE,YLINE,GRADIENT,STDDEV,R2
      DOUBLE PRECISION X,Y
      DIMENSION X(100),Y(100)
      INTEGER L,N
      WRITE(*,*)
      WRITE(*,'('' LINEAR REGRESSION
PROGRAM'')')
      WRITE(*,*)
      WRITE(*,'('' OCTOBER 1998'')')
      WRITE(*,*)
      WRITE(*,'('' VERSION 1.02.'')')
      WRITE(*,*)
      WRITE(*,'('' THIS PROGRAM WILL CALCULATE
THE LINE OF BEST'',
     1/ ,'' FIT FOR UPTO 100 DATA POINTS.'')')
      WRITE(*,*)
      WRITE(*,'('' LETS BEGIN!'')')
      WRITE(*,*)
      DO 1 L=1,100
      WRITE(*,'('' ENTER X,Y (OR -999 ON X TO
START) : '',$)')
      READ *,X(L),Y(L)
      IF (X(L).EQ.-999.) GOTO 2
    1 CONTINUE
    2 N=L-1
      WRITE(*,*)
      WRITE(*,'('' OKAY, I WILL DO THE MATHS
NOW'')')
      CALL CALCSIGMAS()
      CALL CALCOTHERS()
      WRITE(*,*)
      WRITE(*,'('' ANSWERS : '')')
      WRITE(*,*)
      PRINT *,'INTERCEPT ON X ',XLINE
      PRINT *,'INTERCEPT ON Y ',YLINE
      PRINT *,'GRADIENT       ',GRADIENT
      PRINT *,'STANDARD DEVN  ',STDDEV
      PRINT *,'R SQUARED      ',R2
      END

C SUBROUTINE CALCULATE SIGMAS
C CALCULATES ALL THE BITS REQUIRED FOR THE
MAIN PROGRAM
      SUBROUTINE CALCSIGMAS()
      COMMON/SETUP/X,Y,N

COMMON/SIGMAS/SIGMAX,SIGMAX2,SIGMAY,SIGMAX22,S
IGMAXY,AVEX,AVEY
      INTEGER LL,N
      DOUBLE PRECISION X(100),Y(100)
      DOUBLE PRECISION
SIGMAX,SIGMAX2,SIGMAY,SIGMAX22,AVEX,AVEY,SIGMA
XY
      DOUBLE PRECISION SX,SY,SXY,SX2
C SIGMAX22 IS THE SAME AS (SIGMA X)^2
      SX=0.D0
      SY=0.D0
      SXY=0.D0
      SX2=0.D0
      DO 1 LL=1,N
      SIGMAX=SX+X(LL)
      SIGMAY=SY+Y(LL)
      SIGMAXY=SXY+(X(LL)*Y(LL))
      SIGMAX2=SX2+(X(LL)**2)
      SX=SIGMAX
      SY=SIGMAY
      SXY=SIGMAXY
      SX2=SIGMAX2
    1 CONTINUE
      SIGMAX22=SIGMAX**2
      AVEX=SIGMAX/N
      AVEY=SIGMAY/N
      RETURN
      END

C SUBROUTINE CALCULATE THE OTHERS
C CALCULATES ALL THE USEFUL BITS SUCH AS THE
GRADIENTS ETC
      SUBROUTINE CALCOTHERS()
      COMMON/SETUP/X,Y,N

COMMON/SIGMAS/SIGMAX,SIGMAX2,SIGMAY,SIGMAX22,S
IGMAXY,AVEX,AVEY

COMMON/BIGGIES/XLINE,YLINE,GRADIENT,STDDEV,R2
      INTEGER LL,N
      DOUBLE PRECISION
XLINE,YLINE,GRADIENT,STDDEV,R2
      DOUBLE PRECISION X(100),Y(100)
      DOUBLE PRECISION
SIGMAX,SIGMAX2,SIGMAY,SIGMAX22,AVEX,AVEY,SIGMA
XY
      DOUBLE PRECISION M,C,S,YEYC2,YEYC
      DOUBLE PRECISION
XX,SIGMAXXBYYB,RDBIT1,RDBIT2
      DIMENSION
YEYC(100),XXBYYB(100),XXB(100),YYB(100)
      DOUBLE PRECISION XXBYYB,XXB,YYB
C OKAY, LET'S DO THE GRADIENT
      M=(SIGMAX*SIGMAY)-(N*SIGMAXY)
      GRADIENT=M/(SIGMAX22-(N*SIGMAX2))
C INTERCEPT ON Y
      C=(SIGMAX*SIGMAXY)-(SIGMAY*SIGMAX2)
      YLINE=C/(SIGMAX22-(N*SIGMAX2))
C INTERCEPT ON X
      XLINE=-YLINE/GRADIENT
C THESE ARE THE BRUTES!
C STANDARD DEVIATION
      M=GRADIENT
      C=YLINE
      YEYC2=0
      XX=XLINE
      DO 1 LL=1,N
      YEYC(LL)=Y(LL)-(M*X(LL)+C)
```

```
        YEYC(LL)=YEYC(LL)**2
      1 CONTINUE
        DO 2 LL=1,N
        YEYC2=YEYC2+YEYC(LL)
      2 CONTINUE
        S=YEYC2/(N-2.)
        STDDEV=DSQRT(S)
C R^2
        DO 3 LL=1,N
        XXB(LL)=X(LL)-AVEX
        YYB(LL)=Y(LL)-AVEY
        XXBYYB(LL)=XXB(LL)*YYB(LL)
      3 CONTINUE
        SIGMAXXBYYB=0
        DO 4 LL=1,N
        SIGMAXXBYYB=SIGMAXXBYYB+XXBYYB(LL)
      4 CONTINUE
        RDBIT1=0.D0
        RDBIT2=0.D0
        DO 5 LL=1,N
        RDBIT1=RDBIT1+XXB(LL)**2
        RDBIT2=RDBIT2+YYB(LL)**2
      5 CONTINUE
        R2=SIGMAXXBYYB/DSQRT(RDBIT1*RDBIT2)
        IF(R2.LT.0.) R2=R2*(-1)
        RETURN
        END
```

On the surface, this is quite a simple piece of code which should not cause too many problems. Some aspects though are not that easy to convert over to C without knowledge of how much of how F77 works.

```
COMMON/SETUP/X,Y,N
COMMON/SIGMAS/SIGMAX,SIGMAX2,SIGMAY,SIGMAX22,S
IGMAXY,AVEX,AVEY
COMMON/BIGGIES/XLINE,YLINE,GRADIENT,STDDEV,R2
```

`COMMON` works in a very similar way to storage on the heap with an identifier and some variable names, all separated with a `/`. The problem comes in that the storage types are not fixed (`X` is both a `DOUBLE PRECISSION` as well as an array – the method of declaration is similar to that used in C89 whereby a typical prototype would be `int foo(a, b) double a, int b { ... }`) and also the types can differ within the `COMMON` block (`X`, `Y` and `N` do not all need to be of the same type).

The main problem occurs when passing values between functions. In C, this can be achieved by either passing by value or by placing data on the stack via a global `struct`. Global variables can also be used, but this then has a performance overhead.

Use of f2c

The use of f2c is a happy medium. It rewrites the F77 source automatically and links to its own library and any standard C compiler should be able to handle the code as long as the f2c library is available.

Using f2c has problems though

- The code generated, while readable, doesn't make very much sense and without knowing what the f2c functions do, the code may not be maintainable.
- It requires compilation for the target platform and the libf2c dynamic library needs to be shipped to any co-researchers and again, if they are not operating on the same platform, they will need to compile libf2c for their own system.
- There is an increase in both source size and final binary. This does not greatly affect the operating speed of the application (based on time trials of native F77, f2c code and hand written C).

Due to these aspects, f2c may provide an alternative, but is it viable?

Sourcecode – Linear Regression – f2c Version

```
/* Linest.f -- translated by f2c*/
#include "f2c.h"
/* Common Block Declarations */

struct {
    doublereal x[100], y[100];
    integer n;
} setup_;

#define setup_1 setup_

union {
    struct {
   doublereal sigmax, sigmax2, sigmay,
sigmax22, avex, avey;
    } _1;
    struct {
   doublereal sigmax, sigmax2, sigmay,
sigmax22, sigmaxy, avex, avey;
    } _2;
} sigmas_;

#define sigmas_1 (sigmas_._1)
#define sigmas_2 (sigmas_._2)

struct {
    doublereal xline, yline, gradient, stddev,
r2;
} biggies_;

#define biggies_1 biggies_

/* Table of constant values */

static integer c__5 = 5;
static integer c__1 = 1;
static integer c__9 = 9;

/* Linear regression utility */
/* Main program */ int MAIN__(void)
{
    /* Builtin functions */
    integer s_wsle(cilist *), e_wsle(void),
s_wsfe(cilist *), e_wsfe(void),
       s_rsle(cilist *), do_lio(integer *,
integer *, char *, ftnlen),
       e_rsle(void);

    /* Local variables */
    static integer l;
    extern /* Subroutine */ int
calcsigmas_(void), calcothers_(void);

    /* Fortran I/O blocks */
    static cilist io___1 = { 0, 6, 0, 0, 0 };
    static cilist io___2 = { 0, 6, 0, "('
LINEAR REGRESSION PROGRAM')", 0 };
    static cilist io___3 = { 0, 6, 0, 0, 0 };
    static cilist io___4 = { 0, 6, 0, "('
OCTOBER 1998')", 0 };
    static cilist io___5 = { 0, 6, 0, 0, 0 };
    static cilist io___6 = { 0, 6, 0, "('
VERSION 1.02')", 0 };
    static cilist io___7 = { 0, 6, 0, 0, 0 };
    static cilist io___8 = { 0, 6, 0, "(' THIS
PROGRAM WILL CALCULATE THE LI"
      "NE OF BEST',      / ,' FIT FOR UPTO 100
DATA POINTS.')", 0 };
    static cilist io___9 = { 0, 6, 0, 0, 0 };
    static cilist io___10 = { 0, 6, 0, "('
LETS BEGIN!')", 0 };
    static cilist io___11 = { 0, 6, 0, 0, 0 };
    static cilist io___13 = { 0, 6, 0, "('
ENTER X,Y (OR -999 ON X TO START)"
      " : ',$)", 0 };
    static cilist io___14 = { 0, 5, 0, 0, 0 };
    static cilist io___15 = { 0, 6, 0, 0, 0 };
    static cilist io___16 = { 0, 6, 0, "('
OKAY, I WILL DO THE MATHS NOW')",
```

```
       0 };
    static cilist io___17 = { 0, 6, 0, 0, 0 };
    static cilist io___18 = { 0, 6, 0, "('
ANSWERS : ')", 0 };
    static cilist io___19 = { 0, 6, 0, 0, 0 };
    static cilist io___20 = { 0, 6, 0, 0, 0 };
    static cilist io___21 = { 0, 6, 0, 0, 0 };
    static cilist io___22 = { 0, 6, 0, 0, 0 };
    static cilist io___23 = { 0, 6, 0, 0, 0 };
    static cilist io___24 = { 0, 6, 0, 0, 0 };



    s_wsle(&io___1);
    e_wsle();
    s_wsfe(&io___2);
    e_wsfe();
    s_wsle(&io___3);
    e_wsle();
    s_wsfe(&io___4);
    e_wsfe();
    s_wsle(&io___5);
    e_wsle();
    s_wsfe(&io___6);
    e_wsfe();
    s_wsle(&io___7);
    e_wsle();
    s_wsfe(&io___8);
    e_wsfe();
    s_wsle(&io___9);
    e_wsle();
    s_wsfe(&io___10);
    e_wsfe();
    s_wsle(&io___11);
    e_wsle();
    for (l = 1; l <= 100; ++l) {
  s_wsfe(&io___13);
  e_wsfe();
  s_rsle(&io___14);
  do_lio(&c__5, &c__1, (char *)&setup_1.x[l -
1], (ftnlen)sizeof(
     doublereal));
  do_lio(&c__5, &c__1, (char *)&setup_1.y[l -
1], (ftnlen)sizeof(
     doublereal));
  e_rsle();
  if (setup_1.x[l - 1] == -999.f) {
      goto L2;
  }
/* L1: */
    }
L2:
    setup_1.n = l - 1;
    s_wsle(&io___15);
    e_wsle();
    s_wsfe(&io___16);
    e_wsfe();
    calcsigmas_();
    calcothers_();
    s_wsle(&io___17);
    e_wsle();
    s_wsfe(&io___18);
    e_wsfe();
    s_wsle(&io___19);
    e_wsle();
    s_wsle(&io___20);
    do_lio(&c__9, &c__1, "INTERCEPT ON X ",
(ftnlen)15);
    do_lio(&c__5, &c__1, (char
*)&biggies_1.xline,
(ftnlen)sizeof(doublereal))
      ;
    e_wsle();
    s_wsle(&io___21);
    do_lio(&c__9, &c__1, "INTERCEPT ON Y ",
(ftnlen)15);
    do_lio(&c__5, &c__1, (char
*)&biggies_1.yline,
(ftnlen)sizeof(doublereal))
      ;
    e_wsle();
    s_wsle(&io___22);
    do_lio(&c__9, &c__1, "GRADIENT       ",
(ftnlen)15);
    do_lio(&c__5, &c__1, (char
*)&biggies_1.gradient, (ftnlen)sizeof(
      doublereal));
    e_wsle();
    s_wsle(&io___23);
    do_lio(&c__9, &c__1, "STANDARD DEVN  ",
(ftnlen)15);
    do_lio(&c__5, &c__1, (char
*)&biggies_1.stddev,
(ftnlen)sizeof(doublereal)
      );
    e_wsle();
    s_wsle(&io___24);
    do_lio(&c__9, &c__1, "R SQUARED      ",
(ftnlen)15);
    do_lio(&c__5, &c__1, (char
*)&biggies_1.r2, (ftnlen)sizeof(doublereal));
    e_wsle();
    return 0;
} /* MAIN__ */

/* SUBROUTINE CALCULATE SIGMAS */
/* CALCULATES ALL THE BITS REQUIRED FOR THE
MAIN PROGRAM */
/* Subroutine */ int calcsigmas_(void)
{
    /* System generated locals */
    integer i__1;
    doublereal d__1;

    /* Local variables */
    static integer ll;
    static doublereal sx, sy, sx2, sxy;

    sx = 0.;
    sy = 0.;
    sxy = 0.;
    sx2 = 0.;
    i__1 = setup_1.n;
    for (ll = 1; ll <= i__1; ++ll) {
  sigmas_2.sigmax = sx + setup_1.x[ll - 1];
  sigmas_2.sigmay = sy + setup_1.y[ll - 1];
  sigmas_2.sigmaxy = sxy + setup_1.x[ll - 1] *
setup_1.y[ll - 1];
/* Computing 2nd power */
  d__1 = setup_1.x[ll - 1];
  sigmas_2.sigmax2 = sx2 + d__1 * d__1;
  sx = sigmas_2.sigmax;
  sy = sigmas_2.sigmay;
  sxy = sigmas_2.sigmaxy;
  sx2 = sigmas_2.sigmax2;
/* L1: */
    }
/* Computing 2nd power */
    d__1 = sigmas_2.sigmax;
    sigmas_2.sigmax22 = d__1 * d__1;
    sigmas_2.avex = sigmas_2.sigmax /
setup_1.n;
    sigmas_2.avey = sigmas_2.sigmay /
setup_1.n;
    return 0;
} /* calcsigmas_ */

/* SUBROUTINE CALCULATE THE OTHERS */
/* CALCULATES ALL THE USEFUL BITS SUCH AS THE
```

```
GRADIENTS ETC */
/* Subroutine */ int calcothers_(void)
{
    /* System generated locals */
    integer i__1;
    doublereal d__1;

    /* Builtin functions */
    double sqrt(doublereal);

    /* Local variables */
    static doublereal c__, m, s;
    static integer ll;
    static doublereal xx, xxb[100], yyb[100],
sigmaxxbyyb, yeyc[100], yeyc2,
      rdbit1, rdbit2, xxbyyb[100];

/* OKAY, LET'S DO THE GRADIENT */
    m = sigmas_2.sigmax * sigmas_2.sigmay -
setup_1.n * sigmas_2.sigmaxy;
    biggies_1.gradient = m /
(sigmas_2.sigmax22 - setup_1.n *
      sigmas_2.sigmax2);
/* INTERCEPT ON Y */
    c__ = sigmas_2.sigmax * sigmas_2.sigmaxy -
sigmas_2.sigmay *
      sigmas_2.sigmax2;
    biggies_1.yline = c__ / (sigmas_2.sigmax22
- setup_1.n * sigmas_2.sigmax2)
      ;
/* INTERCEPT ON X */
    biggies_1.xline = -biggies_1.yline /
biggies_1.gradient;
/* THESE ARE THE BRUTES! */
/* STANDARD DEVIATION */
    m = biggies_1.gradient;
    c__ = biggies_1.yline;
    yeyc2 = 0.;
    xx = biggies_1.xline;
    i__1 = setup_1.n;
    for (ll = 1; ll <= i__1; ++ll) {
  yeyc[ll - 1] = setup_1.y[ll - 1] - (m *
setup_1.x[ll - 1] + c__);
/* Computing 2nd power */
  d__1 = yeyc[ll - 1];
  yeyc[ll - 1] = d__1 * d__1;
    }
    i__1 = setup_1.n;
    for (ll = 1; ll <= i__1; ++ll) {
  yeyc2 += yeyc[ll - 1];
    }
    s = yeyc2 / (setup_1.n - 2.f);
    biggies_1.stddev = sqrt(s);
/* R^2 */
    i__1 = setup_1.n;
    for (ll = 1; ll <= i__1; ++ll) {
  xxb[ll - 1] = setup_1.x[ll - 1] -
sigmas_2.avex;
  yyb[ll - 1] = setup_1.y[ll - 1] -
sigmas_2.avey;
  xxbyyb[ll - 1] = xxb[ll - 1] * yyb[ll - 1];
    }
    sigmaxxbyyb = 0.;
    i__1 = setup_1.n;
    for (ll = 1; ll <= i__1; ++ll) {
  sigmaxxbyyb += xxbyyb[ll - 1];
    }
    rdbit1 = 0.;
    rdbit2 = 0.;
    i__1 = setup_1.n;
    for (ll = 1; ll <= i__1; ++ll) {
  d__1 = xxb[ll - 1];
  rdbit1 += d__1 * d__1;
  d__1 = yyb[ll - 1];
  rdbit2 += d__1 * d__1;
    }
    biggies_1.r2 = sigmaxxbyyb / sqrt(rdbit1 *
rdbit2);
    if (biggies_1.r2 < 0.f) {
  biggies_1.r2 = -biggies_1.r2;
    }
    return 0;
} /* calcothers_ */

/* Main program alias */ int linreg_ () {
MAIN__ (); return 0; }
```

It is now possible to analyse how f2c converts the F77 to something which resembles C, but is not quite C. The code is understandable to those who use C enabling code to be altered and maintained.

Further analysis though does present problems which may have been overlooked.

- Code is only stored on the heap which is easily corrupted
- Input and output to the program does not make sense
- Errors in the final application may not be down to the binary, but down to f2c

The final alternative is to write your own implementation of the routine in C. As previously discussed though, for anything which is other than trivial (and is not reliant on any third party extension – such as NAG), this will cause difficulties.

Sourcecode – Linear Regression – "pure" C Version

```
#include <stdio.h>
#include <math.h>

struct {
  double x[100];
  double y[100];
  int n;
} data;

struct {
  double sigmax;
  double sigmax2;
  double sigmay;
  double sigmaxy;
  double sigmax22;
  double avex;
  double avey;
} params;

struct {
  double xline;
  double yline;
  double gradient;
  double stddev;
  double r2;
} ans;

void calcsigmas()
{
  double sx = 0, sy  = 0, sxy = 0, sx2 = 0;
  int l;
  for (l = 0; l < data.n; ++l)
    {
      params.sigmax = sx + data.x[l];
      params.sigmay = sy + data.y[l];
      params.sigmaxy = sxy + (data.x[l] *
data.y[l]);
      params.sigmax2 = sx2 + (pow(data.x[l],
2));
      sx = params.sigmax;
      sy = params.sigmay;
      sxy = params.sigmaxy;
      sx2 = params.sigmax2;
    }
  params.sigmax22 = pow(params.sigmax, 2);
```

```
  params.avex = params.sigmax / data.n;
  params.avey = params.sigmay / data.n;
}

void calcothers()
{
  int l;
  double m, c, xx, yeyc2 = 0, sigmaxxbyyb = 0,
rdbit1 = 0, rdbit2 = 0;
  double yeyc[100], xxbyyb[100], xxb[100],
yyb[100];
  m = (params.sigmax * params.sigmay) -
(data.n * params.sigmaxy);
  ans.gradient = m / (params.sigmax22 -
(data.n * params.sigmax2));
  c = (params.sigmax * params.sigmay) -
(params.sigmay * params.sigmax2);
  ans.yline = c / (params.sigmax22 - (data.n *
params.sigmax2));
  ans.xline = -ans.yline / ans.gradient;
  // standard deviation
  m = ans.gradient;
  c = ans.yline;
  xx = ans.xline;
  for (l = 0;l < data.n; ++l)
    {
      yeyc[l] = data.y[l] - (m * data.x[l] +
c);
      yeyc[l] = pow(yeyc[l], 2);
      yeyc2 = yeyc2 + yeyc[l];
    }
  ans.stddev = sqrt(yeyc2 / (data.n - 2));
  // r2
  for (l = 0; l < data.n; ++l)
    {
      xxb[l] = data.x[l] - params.avex;
      yyb[l] = data.y[l] - params.avey;
      xxbyyb[l] = xxb[l] * yyb[l];
      sigmaxxbyyb = sigmaxxbyyb + xxbyyb[l];
      rdbit1 = rdbit1 + pow(xxb[l], 2);
      rdbit2 = rdbit2 + pow(yyb[l], 2);
    }
  ans.r2 = sigmaxxbyyb / sqrt(rdbit1 *
rdbit2);
  if (ans.r2 < 0)
    ans.r2 = ans.r2 * -1;
}

void enterdata()
{
  data.n = 0;
  while (data.n < 100)
    {
      printf("Point : %d. Enter X, Y (or -999
on X to end) : ", data.n);
      scanf("%f%f", &data.x[data.n],
&data.y[data.n]);
      if ((int)data.x[data.n] == -999)
   break;
      data.n++;
    }
}

void printanswers()
{
  printf("Answers\n\n");
  printf("Intercept on x : %f\n", ans.xline);
  printf("Intercept on y : %f\n", ans.yline);
  printf("Gradient       : %f\n",
ans.gradient);
  printf("Standard devn  : %f\n", ans.stddev);
  printf("R squared value: %f\n", ans.r2);
}
```

```
int main()
{
  printf("Linear regression utility - C
version\n");
  enterdata();
  calcsigmas();
  calcothers();
  printanswers();
  printf("Job done\n");
}
```

Again, the data is stored on the heap rather than in a memory block or as discreet global variables. It is, however, far simpler to understand, alter and maintain than the f2c or F77 versions.

Comparison of Source and Binary Sizes

The binary size is based on being linked to dynamic rather than static libraries and were compiled under Red Hat 7.3 Linux using gcc. Sourcecode was written using a simple text editor rather than any form of IDE.

	Binary sizes			Source sizes		
Program	**F77**	**f2c**	**C**	**F77**	**f2c**	**C**
Linest	12.3k	9.9k	7.9k	3.7k	8k	2.7k
Findee9	21.7k	22.8k	18.6k	27.8k	39.4k	17.5k
MOPAC	2Mb	2.3Mb	N/A	1.1Mb	2.0Mb	N/A

MOPAC was not re-written in pure C due to the initial source side and the time involved. It would not be impossible due to the code not requiring any proprietary libraries.

Non-Obvious Problems with f2c

Despite the code being generated from the same code and the generated maths code being virtually identical, compiling and running the binary does show an interesting problem which will directly influence if f2c is even worth bothering with. The data values produced are not the same. This can be demonstrated by entering data into the second test program which generates the elliptic integral of the second kind (incomplete) via the contact and vapour pressures.

It is not a simple matter to find the problem if the generated source is the cause of the problem or the precision used with the standard C libraries. A further complication is that the pure C version of the code gives again a different set of values from both the f2c and F77 versions (the values though were closer to the F77 version than the f2c version which for values closer to the transition point [smaller values] could have been down to rounding errors).

A further factor which would not have been apparent is that for given architecture (ARM as opposed to x86), the results are again different, but still follows the general form shown for x86 machines.

Financial Costs

For trivial code (such as findee9 and linest), the cost of rewriting and testing is negligible and would not be usually factored into the research proposal costings. However, for more complex applications and conversion, the time required would have to factor into the final proposal and justifications made.

For larger scale applications, such as MOPAC, the case for using f2c grows with the additional benefit that smaller sections can be written and tested as part of the current stable code.

Conclusion

From the evidence seen here, the use of f2c, while a useful tool in the regeneration of F77 to C, would not be a suitable alternative for long term purposes. It does have its uses for a quick and dirty replacement, but should not be seen as reason to avoid code rewrites if that is what is required.

Derek Bloor

Reviews

Bookcase

Collated by
<accubooks@progsol.co.uk>

The Editor Writes

It seems that the dropping of the book prices has been met with just about complete acceptance, so I will be carrying on with that from now.

If you want to review a book, your first port of call should be the members section of the ACCU website which contains a list of all of the books currently available. If there is something that you want to review, but can't find on there, just ask. It is possible that we can get hold of it.

You will notice some of the books listed on the website are getting a tad long in the tooth. These are pretty much pointless to review and so the following will happen. After the dates below, books will be dumped.

Those in 2004 will be disposed of on May 1stThose in 2005 will be disposed of on November 1st.

After you've made your choice, email me and if the book checks out on my database, you can have it. I will instruct you from there. Remember though, if the book review is such a stinker as to be awarded the most un-glamourous "not recommended" rating, you are entitled to another book completely free.

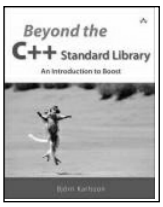
I must thank Blackwells and Computer Bookshop for their continued support in providing us with books.

Paul

The following bookshops actively support ACCU (offering a post free service to UK members – if you ever have a problem with this, please let me know – I can only act on problems that you tell me about). We hope that you will give preference to them. If a bookshop in your area is willing to display ACCU publicity material or otherwise support ACCU, please let us know so they can be added to the list

Computer Manuals (0121 706 6000)
www.computer-manuals.co.uk
Holborn Books Ltd (020 7831 0022)
www.holbornbooks.co.uk
Blackwell's Bookshop, Oxford (01865 792792)
blackwells.extra@blackwell.co.uk

C++



Beyond the C++ Standard Library by Bjorn Karlsson ISBN 0-321-13354-4, Addison-Wesley 388 pp
Reviewed by Francis Glassborow

I expect most readers of this review are already well aware of Boost and the work it has been doing for the last ten years to develop robust and well-designed libraries for C++. Quite a number of these have been further refined and released in the Library Technical Report. Even more of them are likely to be added to the next full release of C++ (due circa 2009).

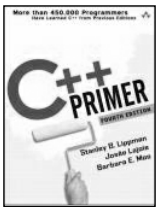
In theory you do not need this book because you can get all the documentation you need from the Web. However, in practice many people like documentation in book form and this book covers 12 of the Boost Libraries. It starts with a chapter on the Smart Pointer library and ends with one on Signals. It covers a good deal of ground in between including chapters on Regex (regular expressions and Lambda.

Each chapter starts with a short piece on how the author thinks this library (the one being covered in the chapter) will improve your programs. It then covers the substance of the library. In each case you will acquire enough knowledge to decide whether you should look at the library in detail be collecting it from the Boost website.

The introduction to the book gives very brief notes on many other Boost libraries.

Unless you are already well familiar with what Boost has to offer, taking time to read this book will be an excellent investment. Unfortunately, there are still software shops who reject all libraries that have not been written in house. We can do little about them except to repeatedly tell them that the result is not a cost effective use of their employees. Others happily use the libraries that come with their development tools, assuming that this will be of high quality, yet still refuse to countenance use of such free libraries as those provided by Boost. The rest of us know that the most thoroughly designed, implemented and tested libraries are those from Boost.

Every self-respecting C++ programmer should be familiar with Boost and have more than a passing knowledge of its major libraries. This book makes acquiring such knowledge easy. If you are not yet familiar with what Boost has to offer, buy this book.



C++ Primer, Fourth Edition by Stanley B. Lippman, Jos?e Lajoie & Barbara E. Moo ISBN 0-201-72148-1, Addison Wesley, 885 pp
Reviewed by Nicola Musatti

Highly recommended.

This is the fourth edition of one of the oldest and most popular introductory books on C++. Since the last edition came out in 1998 C++ as a language hasn't changed much; what has changed is the way the language is used, even at the beginner to intermediate level. The increasing acceptance of the standard library has modified the role of features such as pointers and arrays from fundamental to advanced. This is reflected in the structure of the book, where std::string, std::vector and std::bitset are introduced before null-terminated strings, arrays, pointers and bitfields. These standard library classes are also preferred to their traditional counterparts in most examples, even the very first ones.

The book is opened by an introductory chapter and is then divided in five parts: The Basics, which describes fundamental types and constructs, as well as a few items from the standard library; Containers and Algorithms; Classes and Data Abstraction; Object-Oriented and Generic Programming and Advanced Topics, which presents exception handling, namespaces and other advanced constructs. These are followed by an appendix on the standard library. Each chapter is opened by a short introduction and is closed by a summary and a very convenient glossary of the terms defined within the chapter.

The text is complemented by boxes that contain notes, tips and cautions and by a wealth of exercises for which, however, solutions aren't provided. Material for a companion tome, perhaps.

I found "C++ Primer" well written and easy to understand. It may not be exactly a step-by-step, hand-holding guide but it is certainly suited for people with some programming experience in other languages. I'm convinced that it would also be of use to many C++ programmers, both as a reference and as a means to brush up their knowledge of the language.

I didn't find any major defect in this book. There are a few issues on which I disagree with the authors' point of view, but I have to say that I always found theirs reasonable. This being a first printing it contains a few typographical errors here and there, though not as many as in other recent technical books.

C# and Java

C# for Experienced Programmers by Deitel & Deitel ISBN 1-13-046133-4, Prentice Hall
Review by Paul Thomas

Give it a miss.

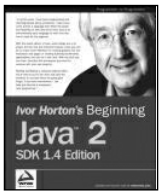
With a title like this, it's not unreasonable to expect that a developer experienced in C++ or Java could learn about the differences fairly quickly and move on to more advanced topics to get a feel for the language. Instead page after tedious page details the "if" keyword or what inheritance is. The pretty diagram inside the back cover recommends novices read their introductory title first. I can't imagine how annoyed I'd be if I had.

To be fair, a Java programmer only really needs an A4 sheet with keyword and class replacement tables. The language was that badly ripped off that it would be hard to pad a thirteen hundred page book like this. This reviewer isn't a Microsoft Hater, but he is a Microsoft sycophant hater. Books like this give you the impression that before the .NET platform, we were all just grubbing around with registers.

There is some useful material contained and it appears to be well researched and checked, the problem is accessing it. Its torturous to read a book like this and the format makes it impossible to simply hunt down the information you need. The over use of bold type reminds me of revising college students that highlight every other sentence in their

notes. Like somehow all of it will be rememebered better.

My other major problem with this book and many like it is that it cannot separate C# from visual studio. This would be fine in a book with a different name but is just misleading here. A more cynical person might think the name was made up purely for product differentiation.



Beginning Java 2 SDK 1.4 Edition by Ivor Horton. ISBN: 0764543652,Wrox Press. 1156 pp.
Reviewed by Frances Buontempo
Highly Recommended

This book is aimed at a Java beginner, though it would also provide good reference or revision for someone with some knowledge of Java programming. For a complete novice, it talks through step by step from the start, though you need to be prepared to read the material slowly and carefully as it is very detailed. If you have no experience of programming this will get you started, giving you a solid foundation provided you work through the examples.

As the cover says, We assume no previous programming experience, although progress will be easier if you have programmed before.? Some readers may find the examples dry, text book stuff, but this provides a solid foundation in the subject.

The book covers working with the basics: numbers and strings, classes, packages, exceptions, streams and files, and collections. It then moves on to a selection of available utility classes, writing graphical user interfaces, concentrating on swing, printing and finally xml.

My only reservation is that it claims to give an ?explanation of Object Oriented programming?. This explanation is not so thorough and the book really just shows how to write classes and use inheritance in Java. If you want to know everything there is to know about OOP, buy a book about that: this book will teach you how to start programming in Java properly, and probably provide an invaluable reference thereafter.

Mac OS X for Java Geeks by Will Iverson ISBN 0-596-00400-1, O'Reilly
Review by Paul Thomas
Highly recommended

For a small book, this covers a surprising amount of material. The main apple extensions are actually quite small, just a few event handlers and utilities, but the book covers everything from the directory structure of the Java framework to setting up application servers. The details of the extension classes has changed since publication, but it's a simple matter to lookup the new class names and the semantics are the same.

Just about everything you could need is here, albeit in an introductory form. The chapter on tools discusses some of the available editors and build tools but I would have liked to have seen more in-depth information on using Xcode (then Project Builder). Later chapters detail converting JAR files to OS X application form or delivering with Java WebStart.

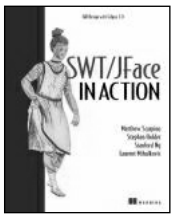
The book deals with more than just Mac peculiarities. It serves as a guide to cross platform development in general. This might sound odd given the "write once, run anywhere" promise of Java, but there's more to it than that. The fashion for Java applications that look the same on all platforms is long gone, and the pluggable look and feel architecture has its own problems. If you develop in a windows environment, you are likely to find that your application looks nasty when dropped onto a Mac.

The basic extension mechanism given is a plugin architecture to isolate the platform specific code. This is used to good effect with a few examples of how an application is integrated into the OS X desktop. Later chapters introduce some of the more interesting APIs such as QuickTime and the Speech API.

The final chapters introduce more enterprise level subjects such as how to set up Tomcat and JBoss. JDBC development is covered with instructions on using MySQL or PostgreSQL. I can't vouch for the accuracy of any of this, but it appears to have been given a clear and concise treatment.

All in all, a very neat little introduction to all things Java on OS X.

Highly recommended if you are a Java developer (of any level) and the Mac is one of your target operating systems.



SWT/JFace in Action byMatthew Scarpino et al ISBN 1-932394-27-3, Manning
Reviewed by Paul Thomas
Recommended

I didn't like the writing style but the layout is excellent. The writing is patchy in the important introductory chapters and it seems unable to decide who the target audience are - sometimes novices, sometimes developers of JFace itself. Each chapter follows that irritatingly popular structure of preview, content, review that student psychologists are so fond of. When well executed, this structure can be effective in guiding students through a subject. Here it has been followed dogmatically. Despite this, the presentation of the content is incredibly clear and easy to follow or reference. The worked examples should be used as examples themselves in how to write programming books.

The content, with the sole exception of the chapter on events, is superb. The treatment of the text widgets in particular does a great job of getting the point across without swamping the reader in details. I think the depth is just about right. Anything not covered in the book can easily be looked up, but those parts that need explaining are covered. Events always seem to be the hardest part to explain.

I never intend to use SWT unless I get sucked into the IBM collective that is Eclipse, but I'm glad I read this book for an alternative view on GUI toolkits. I don't like how SWT is implemented and the use of yet another set of jargon is irresponsible. None of that reflects badly on this book though. Recommended even if you don't need it.

Due to lack of space, not all book reviews could be printed in this issue.
Reviews of the following books are available on the website (www.accu.org) and will be printed in the next issue if space permits.

Programming in the .NET Environment by Damien Watkins, Mark Hammond & Brad Abrams ISBN 0-201-770186-0, Addison-Wesley 523pp Reviewed by Alan Lenton

C# Programmers Cookbook by Allen Jones ISBN 0-7356-1930-1, Microsoft
Reviewed by Paul F. Johnson

Python

Python Essential Reference, 2nd Ed. by David, M. Beasley, ISBN 0-735-71091-0, New Riders.

Python in a Nutshell by Alex Martelli ISBN 0-596-00188-6, O'Reilly.
Reviewed by Ivan Uemlianin

Python and Tkinter Programming by John, E. Grayson ISBN 1-884-77781-3, Manning.
Reviewed by Ivan Uemlianin

Linux

LINUX DEVICE DRIVERS 3E by Jonathon Corbet, Alessandro Rubini & Greg Kroah-Hartman. ISBN 0-596-00590-3, O'Reilly.
Reviewed by Ian Bruntlett.

Software development

eXtreme Programming in Action by Martin Lippert, Stefan Roock, Henning Wolf ISBN 0-470-84705-0, Wiley 232 pp
Reviewed by Joe McCool

Adopting the rational unified process (success with the RUP) by Stefan Bergstr, Lotta R ISBN: 0-321-20294-5, Addison-Wesley
Reviewed by: Michel Greve

The Build Master by Vincent Maraia ISBN 0-321-33205-9, Addison-Wesley 249 pp
Reviewed by Francis Glassborow

The Software Development Edge by Joe Marasco ISBN 0-321-32131-6, Addison-Wesley, 308 pp
Reviewed by Francis Glassborow

Miscellaneous

RFID Field Guide Bhuptani et al. ISBN 0-13-185355-4, Prentice Hall 263 pp
Note by Francis Glassborow

The Man Behind The Microchip - Robert Noyce and the invention of Silicon Valley by Leslie Berlin. ISBN 0-19-516343-5.
Reviewed by Ian Bruntlett